



2004 SEKE

Alberta, Canada

June 20 to June 24, 2004

Proceedings of the
Sixteenth International
Conference on Software
Engineering & Knowledge
Engineering

Sponsored by
Knowledge Systems Institute
iCORE
University of Calgary

PROCEEDINGS

SEKE 2004

The 16th International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Co-Sponsored by

**Informatics Circle of Research Excellence, Canada
University of Calgary, Canada**

Technical Program

June 20-24, 2004

Banff Alberta, Canada

Organized by

Knowledge Systems Institute Graduate School

Copyright © 2004 by Knowledge Systems Institute Graduate School

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the publisher.

ISBN 1-891706-14-4 (paper)

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>

Printed in the United States of America

The 16th International Conference on Software Engineering & Knowledge Engineering (SEKE'2004)

**June 20-24, 2004
Banff, Alberta, Canada**

Organizers & Committee

General Chair

Shi-Kuo Chang, Univ. of Pittsburgh, USA

Program Co-Chairs

Frank Mauer, Associate Head, Department of Computer Science, Univ. of Calgary, Canada
Guenther Ruhe, Industrial Research Chair in Software Engineering, Univ. of Calgary, Canada

Publicity Chair

Jens H. Jahnke, Univ. of Victoria, Canada

Program Committee

Silvia Teresita Acuna, Univ. Autónoma de Madrid, Spain
Anneliese K. Amschler Andrews, Washington State Univ., US
Juan Carlos Augusto, Univ. of Ulster, UK
Aybuke Aurum, Univ. of New South Wales, Australia
Frank Bomarius, Fraunhofer IESE, Germany
Paolo Ciancarini, Univ. of Bologna, Italy
William Chu, Tunghai Univ., Taiwan
John Debenham, Univ. of Technology, Australia
Andrea De Lucia, Univ. of Salerno, Italy
Yi Deng, Florida International Univ., US
Schahram Dustdar, Vienna Univ. of Technology, Austria
Haka Erdogmus, National Research Council Canada, Canada
Filomena Ferrucci, Univ. of Salerno, Italy
Alfonso Fuggetta, Politecnico di Milano Technical Univ., Italy
Carlo Ghezzi, Politecnico di Milano Technical Univ., Italy

Athula Ginige, Univ. of Western Sydney, Australia
Christiane Gresse von Wangenheim, Universidade do Vale do Itajaí, Brazil
Volker Gruhn, Univ. of Leipzig, Germany
John Grundy, Univ. of Auckland, New Zealand
Mao Lin Huang, Univ. of Technology, Australia
Hajimu Iida, Nara Institute of Science and Technology, Japan
Letizia Jaccheri, Norwegian Univ. of Science and Technology, Norway
Natalia Juristo, Technical Univ. of Madrid, Spain
Huimin Lin, Chinese Academy of Sciences, China
Mikael Lindvall, Fraunhofer Center Maryland, US
Jiming Liu, Hong Kong Baptists Univ., China
Luqi, Naval Postgraduate School, US
Sandro Morasca, Univ. degli Studi dell'Insubria, Italy
Jurgen Munch, Fraunhofer IESE, Germany
Lakshmi Narasimhan, Univ. of Newcastle, Australia
Paolo Nesi, Univ. of Florence, Italy
Mehmet Orgun, Macquarie Univ., Australia
Michael Richter, Universitat Kaiserslautern, Germany
Ioana Rus, Fraunhofer IESE, US
Walt Scacchi, Univ. of California Irvine, US
Phillip Sheu, Univ. of California Irvine, US
Eleni Stroulia, Univ. of Alberta, Canada
Scott Tilley, Florida Institute of Technology, US
Genny Tortora, Univ. of Salerno, Italy
Jeffrey Tsai, Univ. of California Irvine, US
Sira Vegas, Universidad Politécnica de Madrid, Spain
Giuseppe Visaggio, Univ. of Bari, Italy
Giuliana Vitiello, Univ. of Salerno, Italy
Yingxu Wang, Univ. of Calgary, Canada
Stefan Wermter, Univ. of Sunderland, UK
Xindong Wu, Univ. of Vermont, US
Yiyu Yao, Univ. of Regina, Canada
Kang Zhang, Univ. of Texas, US

Proceedings Cover Designer

Gabriel Smith, Knowledge Systems Institute Graduate School, USA

Conference Secretariat

Judy Pan, Chair, Knowledge Systems Institute Graduate School, USA
Tony Gong, Knowledge Systems Institute Graduate School, USA
C. C. Huang, Knowledge Systems Institute Graduate School, USA
Beverly Huggins, Knowledge Systems Institute Graduate School, USA
Rex Lee, Knowledge Systems Institute Graduate School, USA
Daniel Li, Knowledge Systems Institute Graduate School, USA

International Workshop on Knowledge Oriented Maintenance (KOM'2004)

**June 20, 2004
Banff, Alberta, Canada**

Organizers

Nicolas Anquetil, Department of Knowledge Management and Information Technology (MGCTI), Catholic Univ. of Brasília, Brazil

Timothy C. Lethbridge, School of Information Technology and Engineering, Univ. of Ottawa, Canada

Program Committee

Nicolas Anquetil, Brazil

Giuliano Antoniol, Italy

Françoise Balmas, France

Dirk Deridder, Belgium

Nicolas Gold, United Kingdom

Idris H. His, U.S.A.

Timothy C. Lethbridge, Canada

Andrea de Lucia, Italy

Ettore Merlo, Canada

Rosângela Ap. Dellosso Penteado, Brazil

Eleni Stroulia, Canada

International Workshop on Ontology In Action (OIA'2004)

**June 21, 2004
Banff, Alberta, Canada**

Workshop Co-Chairs

Athula Ginige, School of Computing and Information Technology, Univ. of Western Sydney,
Australia
Káthia Marçal de Oliveira, Catholic Univ. of Brasília, Brasília, DF, Brazil

Program Committee

Bernhard Holtkamp, Fraunhofer Institut Software-und Systemtechnik, Germany
Ricardo de Almeida Falbo, Federal Univ. of Espírito Santo, Brazil
John Domingue, Knowledge Media Institute, The Open Univ., UK
Uma Srinivasan, CSIRO ICT Centre, Australia
Steffen Staab, Univ. of Karlsruhe, Germany
Kerry Taylor, CSIRO ICT Centre, Australia
Giancarlo Guizzardi, Univ. of Twente, The Netherlands
Luigi Ceccaroni, Univ. Politecnica de Catalunya, Spain
Germana Meneses da Nóbrega, Catholic Univ. of Brasília, Brazil
Mike Uschold, The Boeing Company, Seattle, USA

3rd International Workshop on Software Engineering Decision Support (SEDECS'2004)

**June 22, 2004
Banff, Alberta, Canada**

Program Chair

Guenther Ruhe, Industrial Research Chair in Software Engineering, Univ. of Calgary, Canada

Program Committee

Stephan Biffl, Austria
Khaled El-Emam, Canada
Ross Jeffery, Australia
Sandro Morasca, Italy
Dietmar Pfahl, Germany
David Raffo, USA
Iona Rus, USA
Claes Wohlin, Sweden

Foreword

The Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2004) is held at the Banff Centre, Banff, Alberta, Canada from June 20 to June 24, 2004. The conference brings together experts in software engineering and knowledge engineering to discuss relevant results in both disciplines. Special emphasis is put on synergies between both domains. The conference received nearly 150 technical papers. After a detailed review process, 38% of the submissions were accepted as long papers and an additional 17% as short presentations. Long papers were accepted based on their research quality while short papers and workshop submissions usually report on research in progress and new ideas.

The conference presentations cover a wide spectrum of software engineering and knowledge engineering topics including software processes and process improvement, experience management, quality assurance & testing, requirements engineering, decision support and fuzzy SE knowledge, web engineering, ontologies and agent technology, design and patterns, and formal specification. Authors provide new insights and perspectives on future research directions. The papers included in the conference proceedings speak for themselves.

Several workshops are running in addition to the main conference. The Canadian Agile Network invited to take part in its Second Canadian Agile Network Workshop. The goal of the workshop is to disseminate ideas, lessons learned and best practices of adopting agile methods and moving them to the mainstream of software development. This year the main focus is on agile culture, following organizational change and agile project management.

The Workshop on Knowledge-oriented Maintenance investigates the role of “knowledge” on software maintenance process. Participants share their experience on the extraction and application of knowledge in software maintenance processes.

The Workshop on Learning Software Organizations (LSO 2004) brings together researchers from industry and academia to discuss how continuous learning processes can be implemented and supported in software development teams. Its focus is on practical applications and experience reports.

The Workshop “Ontology in Action” elaborates how shared ontologies can be formalized and used for sharing information amongst heterogeneous software applications. It focuses on how semantic interoperability can be reached by modeling entities and their relationships as domain ontologies.

The Workshop on Software Engineering Decision Support is devoted to discuss methodology, tools and experience on providing support for decision-making in software development and evolution.

We are grateful to all the members of the Program Committee: Silvia Teresita Acuna, Anneliese K. Amschler Andrews, Juan Carlos Augusto, Aybuke Aurum, Frank Bomarius, Paolo Ciancarini, William Chu, John Debenham, Andrea De Lucia, Yi Deng, Schahram Dustdar, Haka Erdogmus, Filomena Ferrucci, Alfonso Fuggetta, Carlo Ghezzi, Athula Ginige, Christiane Gresse von Wangenheim, Volker Gruhn, John Grundy, Mao Lin Huang, Hajimu Iida, Letizia Jaccheri, Natalia Juristo, Huimin Lin, Mikael Lindvall, Jiming Liu, Luqi, Sandro Morasca, Juergen Muench, Lakshmi Narasimhan, Paolo Nesi, Mehmet Orgun, Michael Richter, Ioana Rus, Walt Scacchi, Phillip Sheu, Eleni Stroulia, Scott Tilley, Genny Tortora, Jeffrey Tsai, Sira Vegas, Giuseppe Visaggio, Giuliana Vitiello, Yingxu Wang, Stefan Wermter, Xindong Wu, Yiyu Yao, Kang Zhang. The program committee did an enormous job to review a large number of submitted papers. Their effort ensured the final quality of the conference and all the workshops.

In addition to our program committee members, we would like to thank the following reviewers for providing feedback on submitted papers: Piefrancesco Bellini, Sami Beydeda, Alessandro Bianchi, Kai-Yuan Cai, Zhining Cao, Rosa M. Carro, Alejandra Cechich, María Dolores Vargas Cerdán, Yurong Chen, Oscar Corcho, Patricia Costa, Feras T. Dabous, Angélica de Antonio, Vincenzo Deufemia, Oscar Dieste, Paolo Donzelli, Toncan Duong, Pascal Fenkam, Xavier Ferre, Andres Flores, Rita Francese, Shu Gao, Marisol Giardina, Haitao Gong, Carmine Gravino, Thomas Gschwind, Mariele Hagen, Aaron Hector, Bayu Hendradjaya, Pilar Herrero, Lorin Hochstein, Siv Hilde Houmb, Hiroshi Igaki, José Antonio Macías Iglesias, Zhi Jin, Kanta Jiwnani, André Köhler, Jun Kong, Serguei Krivov, Cat Kutay, Guojun Li, Jingzhou Li, Sheldon X. Liang, Hong-Xin Lin, Pdero Linares, Fabiola Lopez y Lopez, Sergio Di Martino, Nelson Medinlla, Gonzalo Méndez, Abdallah Mohamed, Ana M. Moreno, Ming Muo, Abhaya Nayak, Josef Nedstam, An Ngo-The, Andrew O'Fallon, Alvaro Ortigosa, Thomas Østerlie, Luca Paolino, Orest Pilskalns, Martin Pinzger, Giuseppe Polese, Yu Qian, Fethi Rabhi, Jaime Ramírez, Michele Risi, Omolade Saliu, Marisa Sanchez, Maria-Isabel Sanchez-Segura, Maribel Sanchez-Segura, Giuseppe Scanniello, Klaus Schmid, Indra Seher, Michele A. Shaw, John Shepherd, Xiaochun Shi, Alejandro Sierra, Almudena Sierra-Alonso, Janice Singer, Guanglei Song, Lorna Stewart, Weixiang Sun, Magne Syrstad, Cora B. Excelente Toledo, Maximiliano Paredes Velasco, Qing Wang, Richard Webber, Ying Yang, JingTao Yao, Huilin Ye, InSeon Yoo, Huiqun Yu, Hairong Yu, Guangcun Zhang, Xu Zhang, Haiyan Zhao, Liming Zhu, Xingquan Zhu

Special thanks to all the sponsors of the conference: The Informatics Circle of Research Excellence (iCORE), the University of Calgary and the Knowledge Systems Institute Graduate School.

Welcome to SEKE'2004!

Frank Maurer & Guenther Ruhe
SEKE 2004 Program Committee Co-Chairs

Table of Contents

Conference Organization	iii
Foreword	viii
 Keynote Papers	
Collecting the Dots	1
<i>Shari Lawrence Pfleeger</i>	
Empirically-Based Software Competences: Synergies between Software & Knowledge Engineering	2
<i>Dieter Rombach, Technical University of Kaiserslautern & Fraunhofer Institute for Experimental Software Engineering</i>	
Web Intelligence, World Knowledge and Fuzzy Logic	3
<i>Lotfi A. Zadeh</i>	
 Panel Position Paper	6
 SEKE Long Papers	
A Methodology for Scenario Development	7
<i>Giuseppe Della Penna, Benedetto Intrigila, Anna Rita Laurenzi and Sergio Orefice</i>	
A Nonparametric Software Reliability Model Based on Kernel Estimator and Optimum Algorithm	13
<i>Han Fengyan, Qin Zheng and Wang Xin</i>	
A Simulation-Based Game for Project Management Experiential Learning	19
<i>Alexandre Dantas, Márcio Barros and Cláudia Werner</i>	
A UML-based Software Engineering Methodology for Agent Factory	25
<i>Rem Collier, Gregory O'Hare and Colm Rooney</i>	
ADAMS: an Artefact-based Process Support System	31
<i>Andrea De Lucia, Fausto Fasano, Rita Francese and Genoveffa Tortora</i>	
Agent Technology Portfolio Manager	37
<i>K. S. Barber, J. Ahn, N. Gujral, D. N. Lam and T. Graser</i>	
AgentService	45
<i>Antonio Boccalatte, Andrea Gozzi, Alberto Grosso and Christian Vecchiola</i>	
An Analytical Framework for Consistency Maintenance Mechanisms in Collaborative Editing Systems	51
<i>Liyin Xue, Mehmet Orgun and Kang Zhang</i>	
An Experience of Fuzzy Linear Regression applied to Effort Estimation	57
<i>Gerardo Canfora, Luigi Cerulo and Luigi Troiano</i>	

An Intensional Tool Applied to French Language Educational Software · · · · ·	62
<i>Honglian (Elena) Li and William W. Wadge</i>	
Analysis of meta-programs: a case study · · · · ·	68
<i>Stan Jarzabek, Shen Ru, Hongyu Zhang and Sun Zhenxin</i>	
Architectural Reflection in Adaptive Systems · · · · ·	74
<i>Francesca Arcelli, Claudia Raibulet, Francesco Tisato and Marzia Adorni</i>	
Automated Assistance for Eliciting User Expectations · · · · ·	80
<i>Orna Raz, Rebecca Buchheit, Mary Shaw, Philip Koopman and Christos Faloutsos</i>	
Automated Support for Knowledge Engineering for A Natural Gas Pipeline Domain · · · · ·	86
<i>Christine W. Chan</i>	
Automatic bug triage using text categorization · · · · ·	92
<i>Davor Cubrani and Gail C. Murphy</i>	
Automatic Mapping of OWL Ontologies into Java · · · · ·	98
<i>Aditya Kalyanpur, Daniel Jiménez Pastor, Steve Battle and Julian Padget</i>	
Black- and White-Box Self-testing COTS Components · · · · ·	104
<i>Sami Beydeda and Volker Gruhn</i>	
Building on-line sales assistance systems with ADVISOR SUITE · · · · ·	110
<i>Dietmar Jannach and Gerold Kreutler</i>	
Case Study Methodology Designed Research in Software Engineering Methodology Validation · · · · ·	117
<i>Seok Won Lee and David C. Rine</i>	
Data-mining in Support of Detecting Class Co-evolution · · · · ·	123
<i>Zhenchang Xing and Eleni Stroulia</i>	
Defining and Qualifying Components in the Design Phase · · · · ·	129
<i>Andrew O’Fallon, Orest Pilskalns, Andrew Knight and Anneliese Andrews</i>	
Digging into the Visitor Pattern · · · · ·	135
<i>Fabian B`uttner, Oliver Radfelder, Arne Lindow and Martin Gogolla</i>	
Document Clustering with Adaptive Term Weighting and Feature Reduction Capabilities · · · · ·	142
<i>T.W. Fox and B.J. Fox</i>	
Effort Estimation for Knowledge-based Configuration Systems · · · · ·	148
<i>A. Felfernig</i>	
Enhancing Mediation Security by Aspect-Oriented Approach · · · · ·	155
<i>Li Yang, Raimund K. Ege and Huiqun Yu</i>	
Enhancing the Message Concept of the Object Constraint Language · · · · ·	161
<i>Stephan Flake</i>	
Entering the Heart of Design: Relationships for Tracing Claim Evolution · · · · ·	167
<i>Shahtab Wahid, C. F. Allgood, C. M. Chewar and D. Scott McCrickard</i>	
Future Proofing and Retargeting Application Logic Using O2XML · · · · ·	173
<i>Marselina Wiharto and Peter Stanski</i>	

GOLD: A Generalized Parsing System ······	179
<i>Devin Cook and Du Zhang</i>	
Grammatically Interpreting Feature Compositions ······	185
<i>Wei Zhao, Barrett R. Bryant, Fei Cao, Rajeev R. Raje, Mikhail Auguston, Carol C. Burt, and Andrew M. Olson</i>	
Information Integration Architecture Development: A Multi-Agent Approach ······	192
<i>Stéphane Faulkner, Manuel Kolp, Tai Nguyen, Adrien Coyette and Tung Do</i>	
Level Construction of Decision Trees in a Partition-based Framework for Classification ······	199
<i>Y.Y. Yao, Y. Zhao and J.T. Yao</i>	
Mapping CM ³ : Upfront Maintenance on CGE&Y's Process Model ······	205
<i>Mira Kajko-Mattsson, Karin Ericsson and Zsofia Szalkai</i>	
Mapping UML Diagrams to a Petri Net Notation for System Simulation ······	213
<i>Zhaoxia Hu and Sol M. Shatz</i>	
Multi-Objective Optimization by CBR GA-Optimizer for Module-Order Modeling ······	220
<i>Taghi M. Khoshgoftaar, Yudong Xiao, and Kehan Gao</i>	
Noise Elimination with Ensemble-Classifer Filtering: A Case-Study in Software Quality Engineering ······	226
<i>Taghi M. Khoshgoftaar and Vedang H. Joshi</i>	
On Modelling an e-shop Application on the Knowledge Level: e-ShopAgent Approach ······	232
<i>Nenad Stojanovic</i>	
Predicting UML Statechart Diagrams Understandability Using Fuzzy Logic-Based Techniques ······	238
<i>José A. Cruz-Lemus, Marcela Genero, José A. Olivas, Francisco P. Romero and Mario Piattini</i>	
Programming ubiquitous software applications: requirements for distributed user interface ······	246
<i>Anders Larsson and Erik Berglund</i>	
Requirements Scenarios Based System-Testing ······	252
<i>Ridha Khedri and Imen Bourguiba</i>	
Reuse of UML Class Diagrams Using Case-Based Composition ······	258
<i>Paulo Gomes, Francisco C. Pereira, Paulo Carreiro, Paulo Paiva, Nuno Seco, Jos L. Ferreira and Carlos Bento</i>	
Reusing Knowledge on Software Quality for Developing Measurement Programs ······	264
<i>Olga Jaufman, Bernd Freimut and Ioana Rus</i>	
Reverse Engineering Software Architecture using Rough Clusters ······	270
<i>J. H. Jahnke and Y. Bychkov</i>	
Software Architecture Modelling and Performance Analysis with Argo/MTE ······	276
<i>Yuhong Cai, John Grundy, John Hosking and Xiaoling Dai</i>	
Software Project Risk Evaluation based on Specific and Systemic Risks ······	282
<i>Hélio R. Costa, Márcio de O. Barros and Guilherme H. Travassos</i>	
Software Traceability via Versioned Hypermedia ······	288
<i>Tien N. Nguyen, Ethan V. Munson and Cheng Thao</i>	

Specification and Validation of Transactional Business Software: An Approach Based on the Exploration of Concrete Scenarios	294
<i>Alexandre Correa and Cláudia Werner</i>	
Specification and Verification of Agent Interaction Protocols	300
<i>Bo Chen and Samira Sadaoui</i>	
Supporting the Requirements Prioritization Process. A Machine Learning approach	306
<i>Paolo Avesani, Cinzia Bazzanella, Anna Perini and Angelo Susi</i>	
Team Tacit Knowledge as a Predictor of Performance in Software Development Teams	312
<i>Sharon Ryan and Rory V O'Connor</i>	
Towards Effectively Appraising Online Stores	318
<i>Ernest Cachia and Mark Micallef</i>	
UCDA: Use Case Driven Development Assistant Tool for Class Model Generation	324
<i>Kalaivani Subramaniam, Dong Liu, Behrouz H. Far and Armin Eberlein</i>	
Using A Scenario Specification Language to Add Context to Design Patterns	330
<i>Reginald L. Hobbs</i>	
Visualizing the evolution of software using softChange	336
<i>Daniel M. German, Abram Hindle and Norman Jordan</i>	

SEKE Short Papers

A Framework for Comprehensive Experience-based Decision Support for Software Engineering Technology Selection	342
<i>Andreas Jedlitschka, Dietmar Pfahl and Frank Bomarius</i>	
Active Connectors for Component-Object based Software Architecture	346
<i>Tahar Khammaci, Adel Smeda, and Mourad Oussalah</i>	
Analyzing Invariant Condition of Running Java Program	350
<i>Theodorus Eric Setiadi, Ken Nakayama, Yoshitake Kobayashi, and Mamoru Maekawa</i>	
Application Semiotics Engineering Process	354
<i>Gang Zhao</i>	
Applying Aspect-Oriented Design in Designing Security Systems: A Case Study	360
<i>Shu Gao, Yi Deng, Huiqun Yu, Xudong He, Konstantin Beznosov and Kendra Cooper</i>	
Applying Ontologies in the KDD Pre-Processing Phase	366
<i>Guillermo Nudelman Hess and Cirano Iochpe</i>	
Automated Risk Assessment for Managing Software Projects	372
<i>B. Ray, T. Klinger, R. Delmonico and P. Santhanam</i>	
Clarifying the Relationship between Software Architecture and Usability	378
<i>Natalia Juristo, Ana M. Moreno and Isabel Sánchez</i>	
Commonality and Requirements Analysis for Mesh Generating Software	384
<i>Spencer Smith and Chien-Hsien Chen</i>	

Contextual Comparison of Discovered Knowledge Patterns	388
<i>A.G. Büchner, M. Baumgarten, J.G. Hughes and W.D. Patterson</i>	
Datawarehouses design: effectivity of the star schema	392
<i>Coral Calero, Manuel Serrano and Mario Piattini</i>	
Distributed Knowledge Based System Using Grid Computing for Real Time Air Traffic Synchronization - ATFMGC	396
<i>Weigang Li, Daniel Amaral Cardoso, Marcos Vinícius Pinheiro Dib, Alba Cristina Magalhães and Alves de Melo</i>	
Extracting Minimal Non-Redundant Implication Rules by Using Quantized Closed Itemset Lattice	402
<i>Yun Li, Zongtian Liu, Wei Cheng, Qiang Wu and Wei Liu</i>	
Formal Description Techniques for CSPs and TCSPs	406
<i>Malek Mouhoub, Samira Sadaoui and Amrudee Sukpan</i>	
Handling unanticipated requirements change with aspects	411
<i>Ana Moreira and João Araújo</i>	
Integrating Security Administration into Software Architectures Design	416
<i>Huiqun Yu, Xudong He, Yi Deng and Lian Mo</i>	
Learning to Select Software Components	421
<i>Valerie Maxville, Chiou Peng Lam and Jocelyn Armarego</i>	
Organizational Knowledge: an XML-based Approach to Support Knowledge Management in Distributed and Heterogeneous Environments	427
<i>Carmen Maidantchik, Gleison Santos and Mariano Montoni</i>	
Sense-and-Respond Grid	431
<i>Jun-Jang Jeng, Henry Chang, J. Chung, J. Schiefer, L. An and L. Zeng</i>	
The KAMET II Architecture for Problem-Solving Method Reuse	435
<i>Osvaldo Cairó and Julio César Alvarez</i>	
Using COSMIC-FFP for Predicting Web Application Development Effort	439
<i>G. Costagliola, F. Ferrucci, C. Gravino, G. Tortora and G. Vitiello</i>	
Web based architecture for Dynamic eCollaborative work	445
<i>Ioakim (Makis) Marmaridis, Jeewani Anupama Ginige and Athula Ginige</i>	

KOM Workshop Papers

Feature Value Propagation Analysis for Natural Language Grammars	449
<i>Ettore Merlo , Michel Gagnon , Giuliano Antoniol and Dominic Letarte</i>	
Recovering Traceability Links between Requirement Artefacts: a Case Study	453
<i>Andrea De Lucia, Fausto Fasano, Rita Francese and Rocco Oliveto</i>	
Reengineering an Industrial Legacy Software Towards an Object-Oriented Knowledge-Based System	457
<i>Hakim Lounis, Kaddour Boukerche and Houari A. Sahraoui</i>	
Towards Knowledge Discovery in Software Repositories to Support Refactoring	462
<i>Jörg Rech</i>	

OIA Workshop Papers

Architecture for an Intelligent Web Application	466
<i>Indra Seher and Athula Ginige</i>	
DMTF - CIM to OWL: A Case Study in Ontology Conversion	470
<i>Dennis Heimbigner</i>	
Experiences in Using a Method for Building Domain Ontologies	474
<i>Ricardo de Almeida Falbo</i>	
Learning Materials Ontology and Semantic Web: a case study in Educational Domain	478
<i>Moysés de Araújo</i>	
Non-taxonomic Relations in Semantic Service Discovery and Composition	482
<i>Michael Lutz</i>	
Supporting Interface Integration with a Simple Ontology	486
<i>Damien Conroy, Jim Buckley and Tony Cahill</i>	
SWETO: Large-Scale Semantic Web Test-bed	490
<i>Boanerges Aleman-Meza, Chris Halaschek, Amit Sheth, I. Budak Arpinar and Gowtham Sannapareddy</i>	
Towards Ontological Modelling of Historical Documents	494
<i>Vanesa Mirzaee, Lee Iverson and Babak Hamidzadeh</i>	

SEDECS Workshop Papers

Applying Evidential Reasoning to Multiple Source Data Integration for Software Engineering Decision Support	498
<i>George Shi, Reda Alhajj and Ken Barker</i>	
Decision Support for Planning Software Evolution with Risk Management	503
<i>D. Greer</i>	
Machine Learning-Based Quality Predictive Models: Towards an Artificial Decision Making System	508
<i>Hakim Lounis and Lynda Ait-Mahiedine</i>	
Requirements for a Tool in Support of SE Technology Selection	513
<i>Andreas Jedlitschka and Dietmar Pfahl</i>	

Reviewers	517
----------------------------	-----

Authors Index	519
--------------------------------	-----

Collecting the Dots

Shari Lawrence Pfleeger, PhD
shari_pfleeger@rand.org
Senior Information Scientist

It is often thought that forestalling major threats such as terrorist attacks or epidemics requires weaving together disconnected pieces of information to reveal broader patterns; in more common terms, we call this "connecting the dots." In this talk, we see that connecting the dots cannot happen unless one takes a prior step: "collecting the dots," that is, bringing scattered pieces of information into some proximity to each other to enable pattern recognition. This presentation discusses the dimensions of solving the problem of "collecting the dots." Any solution involves identifying what information is important and improving its circulation within communities that are in a position to connect the dots so collected. The presentation describes organizational and informational barriers to "collecting the dots" and explores the characteristics of potential technology-supported solutions to overcoming them.

Empirically-Based Software Competences:

Synergies between Software & Knowledge Engineering

Dieter Rombach

Technical University of Kaiserslautern
&
Fraunhofer Institute for Experimental Software Engineering

Systematic predictive software development and maintenance still lacks sufficient knowledge about the effects of individual techniques, methods, and tools regarding their impact on quality, cost and time. Especially, we do not understand the variations of such impact depending on varying human, project and organizational characteristics. For example, little testable evidence exists as to the effects of different testing techniques on reliability for small, medium, or large projects, respectively. Without such knowledge, we cannot expect projects to get better wrt. predictability, nor can we expect to repeat successes across multiple projects with varying characteristics. In our discipline of software engineering, this means we have to establish for all techniques, methods or tools “T” knowledge of the kind “ $G = f(T, C)$ ”, where “G” stands for any aspect of quality, cost or time of interest, “C” stands for all variation characteristics of interest, “f” stands for the correlation between them, and “=” indicates that this relationship is empirical in nature. We call any technique, method or tool “T” a competence wrt. some environment characterized by “C”, if we have enough empirical observations in order to conclude that this relationship is stable within empirical bounds and can be repeated. We call such competence a “law” if the results can be repeated for T, we call it a “theory” if we can manipulate T so that a specific result can be achieved. The biggest challenges include (a) establishing an environment to perform sound empirical studies ranging from controlled to case studies, (b) packaging empirical results into reusable experience (= knowledge based on projects within one’s own organization), and (c) deciding when one has enough empirical studies to establish a law.

In this presentation, I will motivate the concept of a “software engineering competence” as defined above as essential both from a scientific as well as a business perspective. The scientific model requires observation and that such observational results must be testable and challengeable. Common business wisdom requires that business competence is achieved by investing into the key processes in your business process (in our case: development process). Then I will define the concept of software engineering competences in detail, present a process for establishing such competences, and discuss some practical examples. I will then emphasize how software engineering can/should benefit from knowledge engineering techniques for data analysis, organization of experience bases for packaging empirical results in a context-sensitive manner, and decision making for effective reuse. Conversely, I will suggest how knowledge engineering may benefit from software engineering as an application with very specific characteristics. Finally, I will give an overview of a collection of already existing empirical observations, laws and theories (Endres & Rombach: A Handbook for Software and Systems Engineering: Empirical Observations, Laws and Theories; Pearson, 2003), and how they should be used in teaching, research and practice.

Web Intelligence, World Knowledge and Fuzzy Logic

Lotfi A. Zadeh^{*}

In moving further into the age of machine intelligence and automated reasoning, we have reached a point where we can speak, without exaggeration, of systems which have a high machine IQ (MIQ). (Fuzzy Logic, Neural Networks, and Soft Computing, *Communications of the ACM—AI*, Vol. 37, pp. 77-84, 1994). The Web and especially search engines—with Google at the top—fall into this category. In the context of the Web, MIQ becomes Web IQ, or WIQ, for short.

Existing search engines have many remarkable capabilities. However, what is not among them is the deduction capability—the capability to answer a query by a synthesis of information which resides in various parts of the knowledge base. A question-answering system is by definition a system which has this capability. One of the principal goals of Web intelligence is that of evolving search engines into question-answering systems. Achievement of this goal requires a quantum jump in the WIQ of existing search engines.

Can this be done with existing tools such as the Semantic Web and ontology-centered systems--tools which are based on bivalent logic and bivalent-logic-based probability theory? It is beyond question that, in recent years, very impressive progress has been made through the use of such tools. But can we achieve a quantum jump in WIQ? A view which is advanced in the following is that bivalent-logic- based methods have intrinsically limited capability to address complex problems which arise in deduction from information which is pervasively ill-structured, uncertain and imprecise.

The major problem is world knowledge—the kind of knowledge that humans acquire through experience and education. Simple examples of fragments of world knowledge are: Usually it is hard to find parking near the campus in early morning and late afternoon; Berkeley is a friendly city; affordable housing is nonexistent in Palo Alto; almost all professors have a Ph.D. degree; and Switzerland has no ports.

Much of the information which relates to world knowledge-- and especially to underlying probabilities-- is perception-based. Reflecting the bounded ability of sensory organs, and ultimately the brain, to resolve detail and store information, perceptions are intrinsically imprecise. More specifically, perceptions are f-granular in the sense that (a) the boundaries of perceived classes are unsharp; and (b) the values of perceived attributes

^{*} Professor in the Graduate School and Director, Berkeley initiative in Soft Computing (BISC), Computer Science Division and the Electronics Research Laboratory, Department of EECS, University of California, Berkeley, CA 94720-1776; Telephone: 510-642-4959; Fax: 510-642-1712; E-Mail: zadeh@cs.berkeley.edu. Research supported in part by ONR N00014-02-1-0294, ONR N00014-00-1-0621 and the BISC Program of UC Berkeley.

are granular, with a granule being a clump of values drawn together by indistinguishability, similarity, proximity and functionality.

Imprecision of perception-based information is a major obstacle to dealing with world knowledge through the use of methods based on bivalent logic and bivalent-logic-based probability theory. What is needed for this purpose is a collection of tools drawn from fuzzy logic-- a logic in which everything is, or is allowed to be, a matter of degree. The principal tool is Precisiated Natural Language (PNL).

The point of departure in PNL is the assumption that the meaning of a proposition, p , drawn from a natural language, NL, can be represented as a generalized constraint of the form $X \text{ is } R$, where X is the constrained variable; R is the constraining relation; and r is a modal variable, that is, a variable whose value defines the modality of the constraint. The principal modalities are: possibilistic ($r=\text{blank}$); veristic ($r=v$); probabilistic ($r=p$); random set ($r=rs$); fuzzy graph ($r=fg$); usuality ($r=u$); and Pawlak set ($r=ps$). The set of all generalized constraints together with their combinations, qualifications and rules of constraint propagation, constitutes the Generalized Constraint Language (GCL). By construction, GCL is maximally expressive.

A proposition, p , in NL is precisiable if it is translatable into GCL. In this sense, PNL consists of precisiable propositions, with the understanding that not every proposition in NL is precisiable. The importance of PNL derives from the fact that it has a far greater expressive power than predicate-logic-based synthetic languages like LISP, Prolog, SQL, etc. A concept which plays a key role in PNL is that of a protoform—an abbreviation of " prototypical form." Informally, the protoform of a lexical entity such as a proposition, command, question, or scenario is its abstracted summary. For example, the protoform of p : Eva is young, is $A(B) \text{ is } C$, where A is abstraction of age, B is abstraction of Eva, and C is abstraction of young. Similarly, the protoform of p : Most Swedes are tall, is $\text{Count } (B/A) \text{ is } Q$, where A is abstraction of Swedes, B is abstraction of tall Swedes, $\text{Count } (B/A)$ is abstraction of the relative count of tall Swedes among Swedes, and Q is abstraction of most.

The importance of the concept of a protoform derives from the fact that it places in evidence the deep semantic structure of the lexical entity to which it applies. In this sense, propositions p and q are PF-equivalent, written as $\text{PFE}(p, q)$, if they have identical protoforms, that is, identical deep semantic structures. As a simple example, p : Most Swedes are tall, and q : Few professors are rich, are PF-equivalent.

The concept of PF- equivalence serves as a basis for what may be called protoform-centered mode of knowledge organization. In this mode, a protoformal module consists of all propositions which have a specified protoform in common, e.g., $A(B) \text{ is } C$. Submodules of such a module are generated through instantiation of A , B and C . For example, the partially instantiated protoform: price (B) is low, would represent all objects in a universe of discourse, U , whose price is low.

An important function of PNL is that of serving as a deduction language. For this purpose, PNL contains a Deduction Database, DB, which consists of so-called protoformal rules of deduction. Basically, such rules govern generalized constraint propagation, with antecedents and consequents expressed as protoforms. Typically, a protoformal rule of deduction has two parts: symbolic and computational. A simple example is the compositional rule of inference in fuzzy logic. In this case, the symbolic part is: if X is A and (X, Y) is B, then Y is C; and the computational part is: $C = A \circ B$, that is, C is the composition of A and B.

The Deduction Database contains a large number of modules and submodules comprising protoformal rules drawn from a wide range of domains. Examples of such modules are: the Search module, the World Knowledge module, the Extension Principle module, the Probability module, the Possibility module, the Usuality module, etc.

In summary, abandonment of bivalence is a prerequisite to achieving a quantum jump in WIQ. By abandoning bivalence, the door is opened to the use of tools such PNL for adding to search engines two essential capabilities: (a) capability to operate on perception-based information; and (b) question-answering capability. What should be stressed, however, is that achievement of this goal will be a major challenge involving exploration of many new directions.

Panel Discussion: Do we Really Need Support in Software Engineering Decision-Making?

Decisions must be made during all stages of software development and evolution. Decisions on software technologies, processes, resources and tools based are the crystallization points to achieve quality of software-dependent products and services. More effective and more efficient decision support will improve the quality and cost-benefit ratio of decision-making. The impact of better decisions on the quality of software will be all the greater since the focus of the project is on the early stages of the life-cycle.

Computerized decision support should be considered in unstructured decision situations characterized by high complexity, uncertainty, multiple groups with a stake in the decision outcome (multiple stakeholders), large amount of information (especially company data), and/or rapid change in information.

For various types of specific problem during software-lifecycle this support means:

- (i) to facilitate understanding and structuring of the problem under investigations,
- (ii) to understand the information needs for making good decisions,
- (iii) to provide access to information that would otherwise be unavailable or difficult to obtain;
- (iv) to generate solution alternatives,
- (v) to evaluate solution alternatives,
- (vi) to prioritize alternatives by using explicit models that provides structure for particular decisions, and
- (vii) to explain solution alternatives.

Panelists:

Dr. Shari Lawrence-Pfleeger ... see SEKE keynotes

Dr. Dietmar Pfahl is a department head with the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany. He has been project leader of several national and international research and transfer projects with software industry, including organizations such as Bosch Telecom, DaimlerChrysler, Draeger, Ericsson, and Siemens. He has more than 50 refereed publications.

Dr. David Raffo is a Professor of Information Systems and Computer Science at Portland State University. Raffo conducts joint research with leading companies developing models to support software engineering decisions. He has over thirty refereed publications in the field of software engineering and is co-Editor-in-Chief of the international journal of Software Process: Improvement and Practice.

Dr. Guenther Ruhe received a doctorate rer.nat degree in Mathematics with emphasis on Operations Research from Freiberg University, Germany and a doctorate habil.nat. degree from both the Technical University of Leipzig and University of Kaiserslautern, Germany. From 1996 until 2001 he was deputy director of the Fraunhofer Institute for Experimental Software Engineering Fh IESE, Germany. He has an Industrial Research Chair in Software Engineering at University of Calgary and is an iCORE Professor since July 2001.

Mark Stanford

A Methodology for Scenario Development

Giuseppe Della Penna, Benedetto Intrigila, Anna Rita Laurenzi, Sergio Orefice
Dipartimento di Informatica, Università di L'Aquila, Italy
{dellapenna, intrigila, laurenzi, orefice}@di.univaq.it

Abstract

The pervasive use of scenarios in the development of computer systems and software has motivated the need of formalisms for the description and manipulation of scenarios. In this paper we propose a scenario-based methodology to support requirements engineering. This methodology enables to exploit the emerging XML technologies in order to offer powerful ways to create, maintain, distribute and use scenarios.

1. Introduction

Usually, there are many sources of requirements, such as customer information, engineering needs, safety constraints, legislation and product safety. The elicited requirements have to be translated in a more implementation-oriented format, becoming the software requirements of the system being implemented.

In this paper we propose a scenario-based methodology, namely *SMDP* (Scenario Model Development Process), to formalize, manipulate and visualize software requirements. The SMDP methodology consists of five main phases: *scenario definition* (the inception phase, where software requirements are organized in the form of scenarios), *scenario refinement* (the elaboration of more detailed scenarios), *scenario composition* (the composition of different scenarios), *scenario transformation* (the derivation of other forms of specification from scenarios) and *scenario validation* (the consistency and completeness checking on the scenarios).

The use of scenarios for capturing requirements has attracted increasing interest among requirement engineers and the literature on scenario methods, models and notations has proliferated (see Section 3 for a survey of the scenario research area). However, in many scenario-based approaches, the scenario knowledge is often specified using semi-formal methods such as tables, structured text or interaction diagrams. The use of such informal specifications do not exploit many of the potential benefits of scenarios and often introduce inconsistency, ambiguity and redundancy.

Moreover, an informal description of scenarios provides a limited support for representing scenario composition and for modelling dependencies between scenarios. This is a lack, since dependencies between scenarios carry important information about a system and single scenarios are not able to give a global description of an application.

The formalism underlying the SMDP methodology is the *SDML* (Scenario Description Markup Language) language, whose syntax is formally defined through an XML Schema. An early version of SDML can be found in [2]. The SDML formalism has been extended in order to support the composition of scenarios. Moreover, we have developed an editing environment to assist the definition and the refinement of scenarios, a graphical and hypertextual visualization system supporting all the SMDP phases, and a prototype application for the automatic generation of test documents starting from SDML specifications.

2. Scenario Model Development Process

The SMDP methodology is an iterative and incremental process which consists of the following phases: scenario definition, refinement, composition, transformation and validation.

We assume that user requirements have been previously documented or explicitly elicited from the stakeholders. In this way, the SDML user works on an existing user requirements document in order to formalize the software requirements by applying the methodology.

In the following we give an overview of the single phases, describing the tasks accomplished in each of them and their SDML counterpart. We begin with the first three phases that are grouped in a resulting macro-phase called *scenario construction*.

2.1. Scenario Construction

Scenario Definition In the scenario definition phase the software requirements are formally organized in the form of scenarios, conceived as concrete sequences of interactions between the user and a system.

The main tasks accomplished during this phase are the identification of the *actors*, the *items* and the *main scenarios*. In the actors and items identification, the actors and items of the scenario, respectively, are extracted from the informal specification of the problem. Actors are all the active agents (human or otherwise) that interact with the system, whereas items are the objects of the system application domain used by the actors to interact with the system. In the main scenarios identification, the set of *goals* representing the system functionalities are identified. For each goal a main scenario is constructed, which contains a trigger, a set of preconditions, a flow of interactions ending with success and a set of postconditions satisfying that goal.

Using the SDML formalism, all the actors and items are defined in object-oriented structures that formally describe their properties (attributes) and the actions that can be applied on them (methods). All the actors and items that belong to a particular domain are grouped in separate documents and assigned to a *namespace*. The scenarios can then import one or more namespaces and refer to the actors or items they define. This allows to give a precise semantics to each object referred in the scenario and to share the domain knowledge between different scenarios. The SDML language syntax allows to define actor and item namespaces through the `<actorList>` and `<itemList>` elements, respectively.

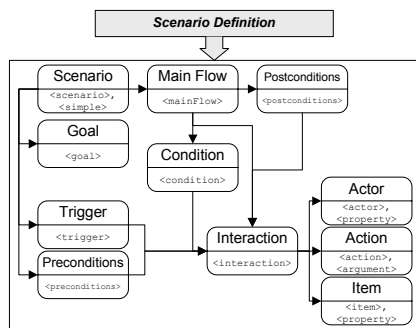


Figure 1. SDML scenario definition.

Figure 1 shows the overall scenario knowledge. The scenario goal is described textually and the interactions are modelled by a main flow containing the scenario knowledge. Such interactions describe the communication between the user and the system through a formal structure in terms of a sequence of actor–action–item, and include the description of the scenario trigger, preconditions and postconditions. Some interactions could be guarded by a particular activating condition.

An interaction is represented using the SDML `<interaction>` element containing the `<actor>`, `<action>` and `<item>` elements. The scenario goal, trigger and the set of preconditions and postconditions have

also a description in the SDML syntax. The goal is represented by the `<goal>` element, whereas the trigger and the preconditions are described through a set of interactions contained in the `<trigger>` and `<preconditions>` elements, respectively. Finally, the success postconditions are also specified as a set of interactions contained in the `<success>` element at the end of the scenario `<mainFlow>`.

To support the definition phase, we have developed a graphical editing environment which assists the SDML user in the construction of scenarios. This editor has been implemented in Java using the Xerces and Xalan packages for the manipulation and transformation of XML, and then results to be highly portable among different platforms.

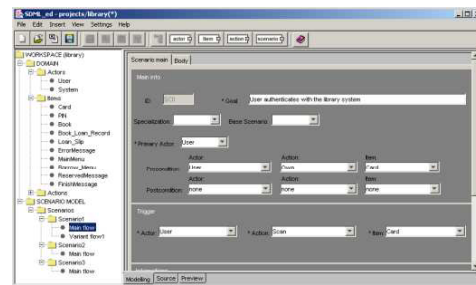


Figure 2. Main interface of the SDML editor.

Figure 2 shows the main interface of such editor. The editor has a left pane that shows the tree-like structure of the scenarios and of the namespaces contained in the current SDML project. Each element in the tree can be clicked to edit the corresponding data. The right pane of the editor window contains a view of the selected element, which can be of different kinds. Typical views consist of a set of wizard-like dialogs that can be used to fill the various SDML structures, or of a text editor where the generated SDML code can be viewed, modified and checked for validity. In Figure 2 the editor is configured in order to enter the scenario information. After filling this form, the user can begin to write the scenario interactions through the “body” section.

We have also developed a visualization system which has been implemented through XSLT stylesheets generating a set of HTML documents animated with *Dynamic HTML*. As an example, Figure 3 shows how the basic structure of a scenario is visualized. The visualization is composed of three parts, showing general information on the scenario (title, author, preconditions, etc), all the namespaces used in the scenario and the scenario interaction flow, respectively. The interaction flow is decomposed as a sequence of steps, each one numbered and associated with a textual explanation, and it always terminates with one the keywords **SUCCESS**, **FAILURE** or **GOTO**.

Scenario SC2121 "Borrow Books"	
Goal	User borrows a book
Primary Actor	User
Secondary Actors	System
Preconditions	User - Own - Card
Trigger	User - Scan - Card
Success	User.borrowed_books > 0
postconditions	System.state = ready
Author	Giuseppe Della Penna
Uses actors from namespace "library"	
Uses items from namespace "library", "general"	
Main Flow FW1	
FW1.1	User - Select_Function (function=Borrow) - Main_Menu
FW1.2	System - Display - Borrow_Menu
FW1.3	User - Scan - Book
FW1.4	System - Print - Loan_Slip
FW1.5	System - Display - FinishMessage
SUCCESS	

Figure 3. Visualization of a scenario.

Scenario Refinement In the scenario refinement phase, scenarios are restructured to make them easy to understand and more reusable. Unlike other works [9] where refinement operations do not increase the contents of scenarios, refinement is used in our approach both to add more details to scenarios (through the notion of *variant flow*) and to abstract common functionalities in order to reuse them in other scenarios (through the notions of *inclusions* and *extensions*).

The main tasks accomplished during this phase are the identification of *redundant flows*, *extension flows*, *failing variant flows* and *alternative flows*. In the identification of redundant flows, sub-flows of interactions common to different scenarios are individuated in order to create new scenarios. These new scenarios will be then recalled in the starting scenarios through appropriate *include rules*. The aim of this task is to remove possible redundancy in the interaction flow through the modularization provided by the inclusion feature. In the identification of extension flows, some new flows of interactions are added to the scenarios through appropriate *extend rules*. This information provides further details to the scenarios without changing their post-conditions. As far as the identification of failing variant flows is concerned, the failure of a scenario means, in general, that its interaction flow does not satisfy the goal. This happens when some condition leads to a ramification of the flow that does not terminate with success. The aim of this task is to identify such kind of conditions and produce the corresponding variant flows. Finally, in the identification of alternative variant flows those interaction flows that may be split in groups of alternative equivalent variant flows are individuated. These flows allow to have more execution paths that satisfy the scenario goal.

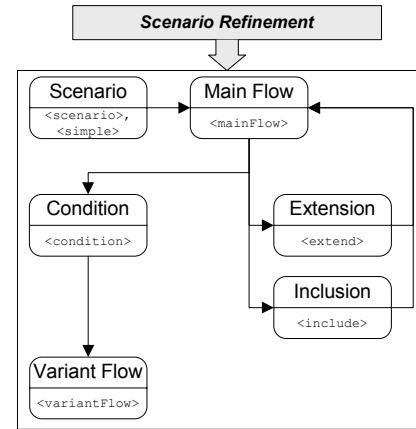


Figure 4. SDML scenario refinement.

Figure 4 shows the overall organization of the SDML scenario refinement elements. A subsequence of interactions in the scenario main flow can be refined by splitting it into different sequences of interactions (failing or alternative variants) that are subject to an activating condition, or by introducing an optional sequence of interactions in a particular point of the main flow as a secondary scenario (extension). Moreover, a scenario can be modularized through groups of sub-flows of interactions that are included in the main flow as new scenarios.

The SDML language syntax includes the element `<variantFlow>` to contain failing or alternative sequences of interactions, and the elements `<extend>` and `<include>` for referring to extensions and inclusions, respectively.

Scenario Composition The role of the scenario composition phase is to compose different scenarios in order to show the dependencies and interactions among the corresponding subsystems. In the SMDP methodology the composition is accomplished directly at the scenario level without introducing other formalisms like it often happens in other approaches.

The main task accomplished in this phase is the identification of relations among scenarios. In general, different scenarios can be related through notions like "sequential", "parallel", "mutually exclusive", "repeated". The aim of this task is to identify all these relationship existing among the involved scenarios and to build a structured composite scenario where they explicitly appear. Thanks to this, it is possible to model features of the system that could not be captured when using a single scenario to describe it.

Figure 5 shows the overall organization of the SDML scenario composition elements. The SDML language syntax provides the notion of *simple* and *composite* sce-

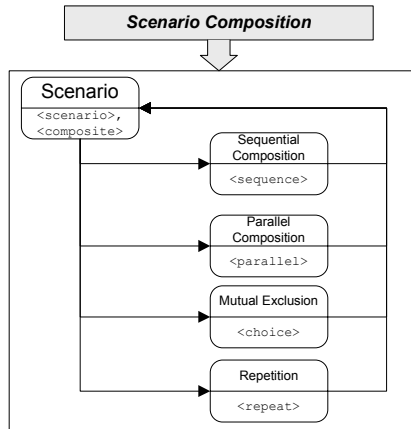


Figure 5. SDML scenario composition.

nario. In the former case, the scenario is contained into a `<simple>` element which describes a main flow with its refinements. In the latter case, the scenario is contained into a `<composite>` element whose content is any valid composition of the `<scenario>`, `<sequence>`, `<parallel>`, `<choice>` and `<repeat>` elements, with the corresponding usual meaning. Note that each of these elements can contain in turn other scenarios, thus producing nested levels of composition.

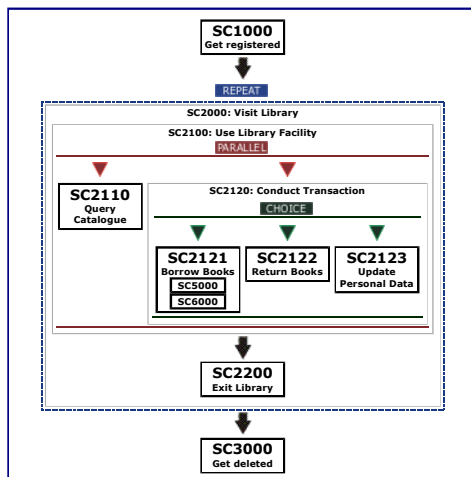


Figure 6. Visualization of the composition in a scenario.

Figure 6 shows how a composite scenario is visualized as a diagram in our visualization system. In the diagram, each component scenario is represented by a box labelled by its identifier. Moreover, whenever a scenario includes other scenarios, these are depicted within the former sce-

nario box. Successive steps of the scenario sequential flow are separated by arrows. Scenarios that must be executed in parallel are visualized on the same row. Depending on the type of parallel composition used, the label *parallel* or *choice* appears over the row. Repeated scenarios are instead visualized inside a dashed box having the label *repeat*.

As said at the beginning of the section, we use the name *scenario construction* to refer to the first three phases of the SMDP methodology described above. The output of the construction phase is the *scenario model*, i.e. a set of scenarios which covers the overall user requirements. The scenario model is built through an incremental process where the requirement coverage gradually increases over iterations among the scenario definition, refinement and composition phases. Each incremental step may generate a new main scenario, refine a scenario adding variants flows, inclusions, extensions, or integrate several scenarios into a composite one.

2.2. Scenario Transformation

In the scenario transformation phase, scenarios are translated in different forms of specification which are used in other phases of the software development process, such as validation and testing, and to produce documentation, too.

In general, this task requires a substantial manual effort and needs the introduction of intermediate formalisms. This is avoided in our approach where the formalization of the scenarios allows to directly translate them into target formats.

As an example, in the following let us describe how our methodology addresses the case of the automatic generation of testing artifacts such as test cases. A test case for the specified system contains a sequence of actions that must be performed during the testing session, followed by the expected system response. Moreover, a test case includes a list of preconditions that must be satisfied before its execution. If the system reacts as expected to the sequence of interactions, then the test is successful.

We have developed a test case generation algorithm that takes a SDML document as input and produces in output a set of test cases by applying three main steps. First of all, the external references (i.e. extensions and inclusions) are resolved in order to build a self-contained scenario. Then, the variant tree is visited and each possible control flow is written separately. These flows are associated to a set of true/false values assigned to the conditional steps they contain, which represent the instance of the preconditions for the test case generated from each flow. Finally, the interactions in each flow are analyzed to generate the corresponding test table. Each test case is then completed by adding the preconditions and other information that are directly copied from the SDML document header. The final

test case is formatted using a standard industrial template and it is expressed in HTML fashion to be easily viewed through a common web browser.

2.3. Scenario Validation

The primary goal of validation is to confirm the elicited requirements and to detect inconsistency, ambiguity and redundancy. The validation should guarantee the correctness and completeness of the requirement specification respect to the user intentions. In general, the validation is said to be *static* if it does not require the execution of the specified software artifact on sample input data, *dynamic* otherwise.

In the SMDP methodology, dynamic checks are accomplished on the target formats generated by the transformation phase, whereas the static verification of some correctness and completeness properties can be directly accomplished on the scenario description.

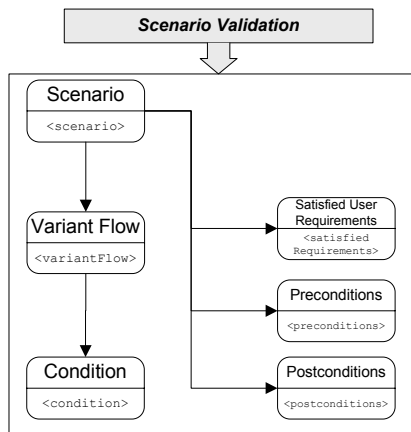


Figure 7. SDML scenario validation.

Figure 7 shows the overall organization of the SDML elements supporting the static validation. Each scenario contains a list of the user requirements that are satisfied through its interaction flow. This allows to establish a relationship from software requirements towards user requirements. Moreover, the preconditions and the postconditions are used to express what the scenario expects to be true when it begins and what it guarantees to be true when it concludes. Finally, each variant contains a list of the conditions that must be true to allow its activation. In this way it is possible, for example, to automatically check if such conditions are satisfied in the specific point of the main flow where that variant appears.

As an example, the elements `<preconditions>`, `<postconditions>` and `<condition>` are used to model the preconditions, the postconditions and the activation conditions of scenarios and variants, respectively.

We have also developed a prototype application written in XSLT based on a meaningful set of quality measures for scenarios, like for example the max depth of the variant tree and the max depth of the failing variant tree. In fact, variants should not be nested too deeply in order to maintain the scenario complexity under an acceptable level.

3. Related Work

In the last years, much research has been done in the scenario-based requirements engineering and a number of approaches has been developed. In the following we give a brief synthesis of some techniques for representing and using scenarios that have been developed so far. A wider survey can be found in [6], where our approach is also more precisely illustrated in comparison with the scenario literature.

In the Potts' et al. approach, [1, 8], scenarios are in textual form following some tabular notations. The requirements engineering process is supported by a hypertext tool in which scenarios and requirements are annotated with requirements discussions, rationales and change requests. Therefore, while inspecting a requirement or a scenario fragment, the user can retrieve, through hypertext links, the open questions, responses and arguments that have been posed on this element and the change requests referring to it as well.

In [4], a concrete style for single scenario representation and a new concept for systematically structuring scenarios and relationships in a set of scenarios are presented. The scenario structure is obtained by combining natural language text with the formal structure of statecharts. Moreover, interaction flow can contain alternatives and iterations.

Sutcliffe et al., [10], define a meta-schema for modeling use cases and scenario based knowledge. The methodology commences by acquisition and modeling of a use case. The use case is then compared with a library of abstract models that represent different application classes, where each model is associated with a set of generic requirements for it class. The authors also provide the CREWS-SAVRE tool to support this methodology.

Leite et al., [7], describe a common scenario construction process and cope with further issues regarding scenario management, in particular the scenario organization. In this approach, scenarios are described in a structured way, using a simple conceptual model together with a form-oriented language.

Hong et al., [5], propose HOONet, a hierarchical object-oriented Petri net, as a method to specify the scenarios and also suggest a technique to integrate scenarios, including different abstraction levels, as well as redundancy, incompleteness and inconsistency.

Only few works in the scenario-based requirements engineering literature exploit the XML technology. Among these, Ralyté, [9], presents an implementation using SGML-HTML to store scenario based approaches in multimedia hypertext documents and illustrates the retrieval of components meeting the requirements of the user by the means of SGMLQL queries.

Duran et al. [3] use XML to represent software requirements and XSLT to support the requirements verification in order to guarantee some quality properties. Nevertheless, scenario representation continues to be semi-formal. For example, the scenario steps are not structured, and the variants miss an activation condition. However, this work is mainly oriented to the implementation and does not aim to the reuse of software specifications.

Not only are scenarios a very relevant research topic, but also they are increasingly adopted in industrial applications. As an example, a survey on the use of scenarios in the software development practice can be found in [11], where fifteen software projects developed using different scenario based approaches are analyzed and compared among them.

4. Concluding remarks

In this paper we presented SMDP, a formal methodology that describes the software specification at various detail levels and allows to reuse the domain knowledge. The SMDP methodology is supported by the underlying SDML formalism, which has been enhanced with a visual editing environment, to create and refine the scenarios, and with a graphical representation system that supports all the SMDP phases and allows to dynamically navigate through the scenario model.

The SMDP methodology has been experimented on a large variety of case studies that have been formalized through the SDML language. For example, some case studies concerning banking or library transactions can be found at the URL <http://dellapenna.univaq.it/sdml/examples.asp>. These experiments have also shown how the SDML editor makes easier the application of the formalism, balancing the difficulties in the learning and the use of SDML due to the high level of formalization adopted.

It is worth noting that each SMDP phase has an underlying XML schema, where a set of appropriate elements has been defined in order to guarantee the traceability among the various phases. Moreover, a specific `<preTraceability>` SDML element supports the traceability of software requirements towards user requirements. More details on these elements can be found in [6].

As further research, we plan to enhance the SMDP validation and transformation phases. In fact, we are currently studying more powerful verification rules to validate scenario models and we are investigating how to derive dif-

ferent formalisms, such as statecharts and SRDs (*software requirements document*), from SDML specifications.

References

- [1] A. I. Anton, W. M. McCracken, and C. Potts. Goal decomposition and scenario analysis in business process reengineering. In *Proceedings of the 6th Conference on Advanced Information Systems Engineering*, pages 94–104, 1994.
- [2] G. Della Penna, B. Intrigila, A. R. Laurenzi, and S. Orefice. An xml definition language to support scenario-based requirements engineering. *International Journal of Software Engineering and Knowledge Engineering*, June 2003.
- [3] A. Duran, A. Ruiz-Cortez, R. Corchuelo, and M. Toro. Supporting requirements verification using xslt. In *Proceedings of IEEE Joint International Conference on on Requirements Engineering (RE02)*, pages 165–172, 2002.
- [4] M. Glinz. An integrated formal model of scenarios based on statecharts. In Springer, editor, *Proceedings of the 5th European Software Engineering Conference*, pages 254–271, 1995.
- [5] Z. Hong and J. Lingzi. Scenario analysis in an automated tool for requirements engineering. *Requirements Engineering Journal*, 5(1):2–22, 2000.
- [6] A. R. Laurenzi. An xml based methodology to model and use scenarios in the software development process. Technical report, Università degli Studi di L'Aquila, 2004.
- [7] J. C. S. P. Leite, G. D. S. Hadad, J. H. Doorn, and G. N. Kaplan. A scenario construction process. *Requirements Engineering Journal*, 5(1):38–61, 2000.
- [8] C. Potts, K. Takahashi, J. Smith, and K. Ora. An evaluation of inquiry based requirements analysis for an internet service. In I. C. Society, editor, *Proceedings of the Second IEEE Symposium on Requirements Engineering*, pages 27–34, 1995.
- [9] J. Ralyté. Reusing scenario based approaches in requirement engineering methods: Crews method base. In I. C. Society, editor, *Proceedings of the 10th International Workshop on Database and Expert Systems Applications (DEXA'99)*, pages 305–309, 1999.
- [10] A. Sutcliffe, N. Maiden, S. Minocha, and D. Manuel. Supporting scenario-based requirements engineering. *IEEE Transactions on Software Engineering*, 24(12), December 1998.
- [11] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer. Scenarios in system development: Current practice. *IEEE Software*, (34-45), March/April 1998.

A Nonparametric Software Reliability Model Based on Kernel Estimator and Optimum Algorithm

Han Fengyan, Qin Zheng, Wang Xin

School of Electronic and Information Engineering, Xian Jiaotong University, Xian China

Email: f.han@263.net

Abstract. *This paper presents a new nonparametric software reliability model that is based on kernel failure rate estimation and optimum algorithm. The core of the model is our nonparametric failure rate estimator based on the weighted kernel function method and under the constraint of failure rate monotonically decreasing. Because of the introduction of a novel weight parameter and an improved distance metric, not only the estimator can converge easily, but also the number of defects of the software system can be estimated. We apply the model to some real data sets and compare the results with that come from some better conventional models. The real data analysis shows that the method performs as well as these better conventional models.*

1. Introduction

It is very important to estimate and predict the reliability of a software system in quality engineering. Up till now, a number of models have been developed for tracking or predicting software reliability. Generally speaking, these models assume that the failure data are drawn from one of a known parametric family of distributions, and then estimate the parameters by least square fit or maximum likelihood estimation^[1]. This implies quite strong constraint. Because of the complexity of software failure, the basic assumptions about these models are sometimes unsound, and often disputed. In 1991, A. Sofer and D. R. Miller^[2] suggested a nonparametric model for software reliability. They assumed that the failure rate is a monotonically decreasing function and then produced valid results, but the calculation is highly dependent on the experience of the analyst.

In order to estimate failure rate of a system, some nonparametric estimators have been developed, among them is the nonparametric estimator based on kernel function, which has been discussed adequately^[3]. And a lot of new nonparametric estimators such as estimator based on neural network and wavelet transform have been developed recently^[4]. To estimate the density and failure rate of failure rate monotonically decreasing system, P. Hall etc^[5] present a weighted kernel method, which involves giving weights to data values, and selecting the weights so as to

minimize the distance function to the uniform case, subject the failure rate to achieving monotonic. The Hall's method works well in correcting the local reverse trend estimated by the unconstrained estimator, but it is difficult to cope with the large extent reverse trend, and mostly it can not be applied to the software system in testing.

In this paper, we present a new nonparametric software reliability model for software system under the assumption of failure rate decreasing monotonically. Adopting the Hall's optimum approach, the model involves a different failure rate estimator, a new weight parameter, an improved distance metric and other extensions, so that the model can be applied to software system in testing. We apply the model to some real data sets and compare the results with that come from conventional models. The results show that the model performs as well as the better conventional models.

2. Model description

Let the initial number of defects of software system under testing is N , and the i th defect is found at T_i , and stop testing when the n th defect is found. Thus we get the data set $\{T_i\} = T_1, T_2, \dots, T_n$ about the survival time (the time to be found in testing) of n defects.

2.1 The basic assumptions

In order to construct the nonparametric software reliability model, we make basic assumptions as following:

Assumption 1. The survival probability of every defect is independent, and obeys the same probability distribution, that is the n defects obey independent identical distribution.

Assumption 2. The defect may be removed when it was found, and no new defect be introduced to the system in the correcting process.

Assumption 3. The failure rate of the software system is decreasing monotonically in the testing process.

2.2 Nonparametric estimation of failure rate

According to assumption1, the survival probability of

every defect is independent and obeys the same probability distribution with density $f(t)$. Then $f(t)$ and corresponding failure rate $\lambda(t)$ can be estimated by kernel function method as following^[3]

$$\hat{f}_1(t) = \frac{1}{nh} \sum_{i=1}^n K[(t - T_i)/h] \quad (1)$$

$$\hat{\lambda}_1(t) = \frac{1}{nh} \sum_{i=1}^n \frac{K[(t - T_i)/h]}{1 - i/N} \quad (2)$$

Where, $K(x)$ is named kernel function, which is a no-negative, symmetry and smooth function supported on $[-L, L]$, and normalized to unity as in equation (3).

$$\int_{-L}^L K(x)dx = 1 \quad (3)$$

The parameter h denotes bandwidth, which determines the smoothing characteristic of the estimated results and is also named smoothing parameter.

We define censored rate β as the ratio of residual number of defects to the initial number of defects

$$\beta = 1 - n/N \quad (4)$$

Then equation (2) leads to

$$\hat{\lambda}_1(t) = \frac{1}{h} \sum_{i=1}^n \frac{K[(t - T_i)/h]}{n - i(1 - \beta)} \quad (5)$$

The equation (5) represents the estimation of failure rate corresponding to an individual defect. Considering the software system with N initial defects, because the defect may be removed immediately when it was found, and no new defect be introduced (assumption2), so after correcting the i th defect, the residual number of defects will be $(N-i)$ in the successive time interval. And thus the failure rate of the software system can be estimated by $(N-i)\hat{\lambda}_1$, that is

$$\hat{\lambda}(t) = \frac{N - I(t)}{h} \sum_{i=1}^n \frac{K[(t - T_i)/h]}{n - i(1 - \beta)} \quad (6)$$

Where $I(t)$ denote the failures counted in $[0, t]$.

In order to obtain estimation of failure rate decreasing monotonically, weights are introduced in kernel function method. Let α_i be the weight corresponding to T_i , and then the failure rate can be estimated by equation (7)

$$\begin{aligned} \hat{\lambda}(\omega, t) &= [N - I(t)] \frac{1}{h} \sum_{i=1}^n \frac{\alpha_i K[(t - T_i)/h]}{n - i(1 - \beta)} \\ &= \left[\frac{n}{1 - \beta} - I(t) \right] \frac{1}{h} \sum_{i=1}^n \frac{\alpha_i K[(t - T_i)/h]}{n - i(1 - \beta)} \end{aligned} \quad (7)$$

And the derivative of failure rate is estimated by equation (8)

$$\begin{aligned} \hat{\lambda}'(\omega, t) &= \left[\frac{n}{1 - \beta} - I(t) \right] \frac{1}{h} \sum_{i=1}^n \frac{\alpha_i K'[(t - T_i)/h]}{n - i(1 - \beta)} \\ &\quad - I'(t) \frac{1}{h} \sum_{i=1}^n \frac{\alpha_i K[(t - T_i)/h]}{n - i(1 - \beta)} \end{aligned} \quad (8)$$

Where, the weights $\omega = [\alpha_1, \alpha_2, \dots, \alpha_n, \beta]$ satisfy

$$0 \leq \omega_i \leq 1, \quad i = 1, 2, \dots, n+1 \quad (9)$$

And ω normalize to unity,

$$\beta + \sum_{i=1}^n \alpha_i = 1 \quad (10)$$

The failure rate decreasing monotonically implies that the derivative of failure rate is no-positive in the estimated interval, that is

$$\hat{\lambda}'(\omega, t) \leq 0 \quad (11)$$

Considering the weights including β , we define the distance between ω and $\bar{\omega} = \frac{1}{n}[1, 1, \dots, 1, 0]$ as

$$\begin{aligned} D_0(\omega) &= - \sum_{i=1}^n \log\left(\frac{n\alpha_i}{1 - \beta}\right) - \log(1 - \beta) \\ &= - \sum_{i=1}^n \log(n\alpha_i) + (n-1) \log(1 - \beta) \end{aligned} \quad (12)$$

If $\beta = 0$, equation (12) will retrogress to the standard distance measure

$$D_0(\alpha) = - \sum_{i=1}^n \log(n\alpha_i) \quad (13)$$

It is worth noticing that the distance measure (12) is no-negative, $D_0(\omega) \geq 0$ for all cases, and $D_0(\omega) = 0$ if and only if $\omega = \bar{\omega}$.

Using equation (7), we can obtain the estimation of failure rate satisfying the monotonically decreasing condition, by selecting weights ω properly to enforce the failure rate decrease in the estimated interval. In another words, we can get the estimation of density and failure rate by solving the optimization problem with objective function $D_0(\omega)$, and under the constraints of equation (9), (10) and (11), that is

$$\left[\begin{array}{l} \min_{\omega} imaze(D_0(\omega)) \\ \text{subject to} \\ \beta + \sum_{i=1}^n \alpha_i = 1 \\ \beta \geq 0, \alpha_i \geq 0, \quad i = 1, 2, \dots, n \\ \hat{\lambda}'(\omega, t) \leq 0, \quad 0 \leq t \leq T_n \end{array} \right. \quad (14)$$

This is a nonlinear optimization problem with nonlinear objective function, and under the constraints of linear equations and nonlinear inequations. The solution of the optimization problem may results density and failure rate with failure rate decreasing monotonically.

2.3 Estimation of initial defects

In the process of solving the optimization problem, an optimized parameter β can be obtained, thus we get the estimated initial number of defects from equation (15).

$$\hat{N} = \frac{n}{1 - \beta} \quad (15)$$

2.4 Prediction of failure rate

To predict the failure rate, we approximate the estimated failure rate by the exponent of a polynomial

$$L(t) = \exp\left(\sum_{j=1}^m a_j t^j\right) \quad (16)$$

and let expression (16) satisfies the condition expressed by equation (17)

$$L'(t_n) = \lambda'(t_n) \quad (17)$$

The coefficients a_j may be determined by least square fit, and thus the future failure rate can be predicted.

3. Real data analysis

3.1 The calculus methods

In the practical calculation, we choose Gaussian function as the kernel function

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \quad (18)$$

which is supported on $(-\infty, +\infty)$.

The estimation results are sensitive to the bandwidth. If h were too small, the curve estimated will not be smooth enough, and will be wave acutely in some range. On the contrary, if h were too large, the curve estimated will be over-smoothed, and the varieties will be difficult to detect. To avoid this difficulty, the adaptive bandwidth is used in the calculation, that is h defined as a local parameter $h_i = k_i h_0$, where h_0 is the so called normal reference bandwidth

$$h_0 = 1.06\sigma \cdot n^{-1/5} \quad (19)$$

Where n is the volume of samples, and σ is the standard deviation of the data.

The coefficient k_i increases with density decreasing, a usual choice of it may be $k_i = (k / f_i)^\gamma$, where γ is another smoothing parameter, and k is the geometric average of the density

$$k = \left(\prod_{j=1}^n f_j \right)^{1/n} \quad (20)$$

In practical calculating, $I(t)$ is redefined as (21) so as to keep the continuity of the failure rate expressed in equation (7)

$$I(t) = i + \frac{t - T_i}{T_{i+1} - T_i} \quad t \in [T_i, T_{i+1}) \quad (21)$$

And its derivative may be expressed as

$$I'(t) = \begin{cases} \frac{1}{T_{i+1} - T_i} & t \in (T_i, T_{i+1}) \\ \frac{2}{T_{i+1} - T_{i-1}} & t = T_i \end{cases} \quad (22)$$

The optimization problem of expression (14) can be solved by sequential quadratic programming (SQP) technique. The SQP method allows you to closely mimic Newton's method for constrained optimization just as what

is done for unconstrained optimization. The principal ideas of SQP method is that, at each major iteration, an approximation is made for the Hessian of the Lagrangian function using a quasi-Newton updating method, thus the nonlinear optimization problem transformed to quadratic programming subproblem, and the quadratic programming subproblem can be solved by general quadratic programming. Using SQP technique we can get the satisfactory solution after enough iterations.

We use u-plot to evaluate the predicative quality of the model. The u-plot is a powerful tool for detecting systematic bias of the model predictions. It is well known that if the random variable T_i truly obeys the distribution $F(t)$, then the random variable $U_i = \hat{F}(T_i)$ would be uniformly distributed on (0,1). Thus when we calculate $u_i = \hat{F}(t_i)$ for a sequence of predictions, we would get a sequence $\{u_i\}$, which looks like a random sample from a uniform distribution. A general method of looking for departure from uniformity is by plotting the sample

distribution function for the $\{u_i\}$ sequence. If the $\{u_i\}$ sequence were truly uniform, this plot should be close to the line of unit slope.

3.2 Results and comparison

We apply the nonparametric model and several better conventional models to the data set of system T_1 (denoted by DT1) presented by Musa^[6], in which the number of defeats detected $n=136$, and the n th defeat was detected at $T_n = 88682$ CPU seconds.

3.2.1 The estimated failure rate

The failure rate estimated by our model and several better conventional models are showed in section I and section II of figure 3.1, where NP stands for our nonparametric model, and JM for Jelinski-Moranda model, GO for Goel-Okumoto model, MO for Musa-Okumoto model, YOO for Yamada-Osaki model.

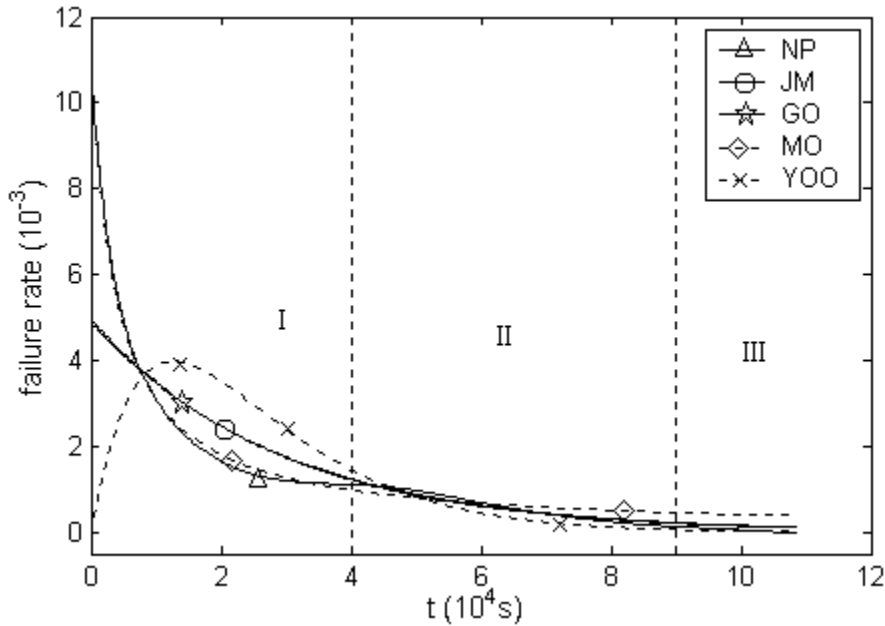


Figure 3.1 Estimated and predicted failure rate

The nonparametric model works well in correcting the upward trend near $t=30000$ s and near the right boundary $t=88682$ s, and produces the failure rate decreasing monotonically.

Compared with the curve estimated by conventional models, in section I (about $t=3\sim 40000$ s), the curve estimated by our nonparametric model coincides approximately with that come from Musa-Okumoto

model, while in section II (about $t=40000\sim 88682$ s), the curve estimated by our nonparametric model coincides with that estimated by Jelinski-Moranda model and the Goel-Okumoto model. And in this section, the curve of nonparametric model is seated between the Musa-Okumoto model and the Yamada-Osaki model.

3.2.2 The predicted failure rate

The long term predictions of failure rate by different models are showed in section III of figure 3.1, which show that the prediction of our nonparametric model is coincident with that of the Jelinski-Moranda model and the Goel-Okumoto model.

The long-term predictions of failure rate by our nonparametric model after $t=88682$ seconds, are listed in table 3.1. The predication may be used to plan the test in the future. For instance, to reach a failure rate of 2.2×10^{-4} one should test the software system for another 15318 seconds.

Table 3.1 Predicted failure rate

t(s)	90000	92000	94000	96000
$\lambda(10^{-3}s^{-1})$	1.33	1.18	1.04	0.91
t(s)	98000	100000	102000	104000
$\lambda(10^{-3}s^{-1})$	0.76	0.59	0.40	0.22

3.2.3 The estimated initial number of defeats

To estimate initial number of defects, we apply the model to censored data subset in DT1. After performing 6 calculations we obtain the estimated initial number of

defects as shown in table 3.2, in which the average of the estimated N is 187.

Table 3.2 Estimated initial number of defects

n	$\hat{\beta}$	\hat{N}	n	$\hat{\beta}$	\hat{N}
131	0.275	180.7	134	0.287	187.8
132	0.293	186.6	135	0.286	189.0
133	0.291	187.5	136	0.279	188.7

Compared with the conventional models, the estimated initial number of defeats is listed in table 3.3, which shows that the estimated values by our nonparametric model are more conservative than other conventional models.

Table 3.3 Comparison of Estimated N

Models	NP	JM	GO	YOO
\hat{N}	187	142	143	137

3.2.4 The u-plot

The u-plot of the nonparametric model and better conventional models for DT1 are showed in figure 3.2.

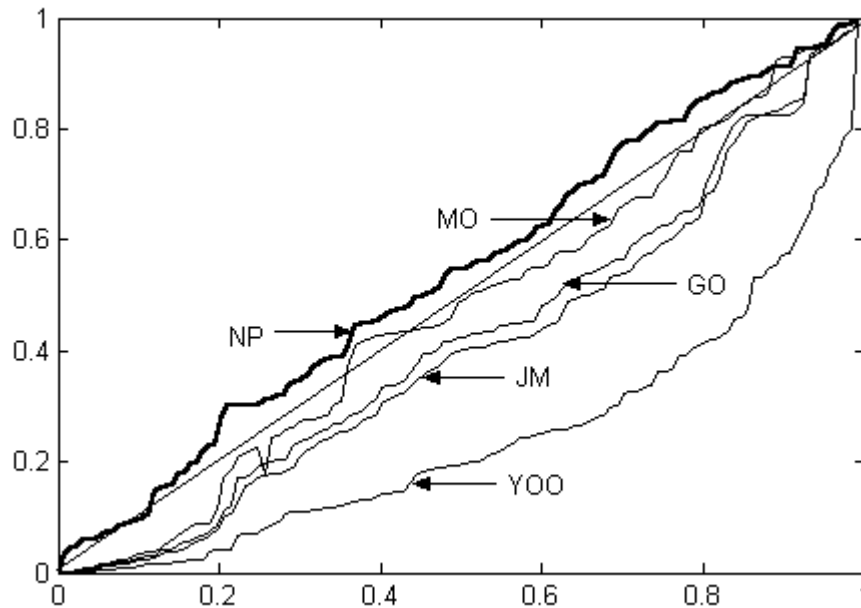


Figure 3.2 The u-plot of DT1

This u-plot reveals that the $u_i = \hat{F}(T_i)$ of nonparametric model is nearly uniformly distributed on (0,1). According to the u-plot, the nonparametric model is very close to Musa-Okumoto model, and is superior to Jelinski-Moranda model, Goel-Okumoto model, and Yamada-Osaki model evidently.

4. Conclusion and discussion

(1) The nonparametric software reliability model presented in the paper can produce reasonable estimation of initial number of defects, and results failure rate decreasing monotonically, which are coincident approximately with the better conventional models. Because the assumptions of the nonparametric model are less strong than that of the better conventional models, the nonparametric model is a more natural approach.

(2) The nonparametric failure rate estimator with parameter α and β can correct the increasing trends effectively. The parameter β may affect on the estimated results entirely, and is especially important for correcting the rapid increase of failure rate in the right boundary.

(3) Not only can this model be applied to failure rate monotonically decreasing system, but also can be applied to failure rate monotonically increasing system theoretically, and hence can be used to test the assumption of increase or decrease monotonically.

(4) Although real data analysis shows that the nonparametric model is applicable in cases that failure rate trends to decrease monotonically, and the model performs as well as better conventional models, but some more theoretical searches, such as believe interval etc, are needed in the future.

References

- [1] M.R. Lyu, Handbook of Software Reliability Engineering, IEEE Computer Society Press, McGraw Hill, 1996.
- [2] A. Sofer, D.R. Miller, a Nonparametric Software Reliability Growth Model, IEEE Transactions on Reliability, VOL40, NO.3, 1991.
- [3] B. W. Silverman, Density Estimation for Statistics and Data Analysis. Monographs on Statistics and Applied Probability. London: Chapman and Hall, 1986.
- [4] D.R.M. Herrick, G.P. Nason, B.W. Silverman, Some New Methods for Wavelet Density Estimation, Technical Report of University of Bristol, 2001.
- [5] P. Hall, L. Huang, J.A. Gifford, I. Gijbels, Nonparametric Estimation of Failure rate Under the Constraint of Monotonicity, Technical Report of Australian National University, 2001. <http://www.citeseer.nj.nec.com/236942.html>.
- [6] Musa, J.D., A. Iannino and K. Okumoto, Software Reliability: Measurement, Prediction, Application, McGraw-Hill, New York, 1987.

A Simulation-Based Game for Project Management Experiential Learning

Alexandre Dantas¹, Márcio Barros^{1,2}, Cláudia Werner¹
{alexrd, marcio, werner}@cos.ufrj.br

¹ COPPE / UFRJ – System Engineering and Computer Science Department
Caixa Postal: 68511 – CEP: 21945-970 – Rio de Janeiro – Brazil

² UNIRIOTEC – Applied Computer Science Department
Av. Pasteur 458, Urca – CEP: 22290-240 – Rio de Janeiro – Brasil

Abstract. The inadequate use of project management techniques in software projects can be traced to the lack of efficient project management education strategies, where learning by experience and motivation are key issues. An experiential learning process for project management requires an environment where students can act as managers without the costs and risks associated to an unsuccessful software project. Simulation can support this process, but simulation tools lack the look-and-feel of a real project development environment. In this paper we propose a simulation-based game that can be used to provide experiential learning to project managers. A System Dynamics model describing a software project, a simulator, and a game machine that handles user interactions and presents simulation results in a game-like fashion compose the game. System Dynamics limitations to support a game-like user interface are discussed. Also, we present an experimental study that evaluates an experiential learning process based on the proposed game.

1. Introduction

Project management can be considered an universal concept, but according to the software engineering literature and recent researches [3][11][16] its adoption in software projects is still inadequate and deficient. The high number of software projects that are cancelled each year and the number of project presenting schedule and cost overruns [3][16] may be consequences of this lack of project management.

It is widely accepted that experienced project managers perform better than inexperienced managers in concluding their projects successfully, that is, within their planned schedules and budgets. Project management is strongly dependent on knowledge and still many project managers are promoted from technical teams due to their

successes in previous projects without proper training and education to acquire management skills [11].

Thus, education strategies adopted to prepare project managers play an important role in preventing from inadequate use of management techniques on software projects, providing the basis to leverage the present scenario of so many faulty projects.

This paper discusses current project management education strategies and their deficiencies. We consider the application of simulation and games to support management training. We have developed a game, namely *The Incredible Manager*, and used it in two experimental studies to evaluate our hypotheses concerning the usefulness of games to a project management training program. This paper presents the game structure and results obtained from the studies.

The paper is organized in eight sections. The first one comprises this introduction. The next section discusses the deficiencies presented by the traditional professor-centric education strategy when applied to project management courses and some research that has been made to complement this approach with other tools. Section 3 presents the game that we have developed and its architecture. The following sections present the major components that compose this architecture: Section 4 presents the simulation model, Section 5 presents the simulation model, while Section 6 describes the game machine. Section 7 discusses the experimental studies that were executed to evaluate the game usefulness. Section 8 concludes the paper by presenting our final considerations.

2. Experience in Education

One of the keys for educational success is motivation and one of the best motivations for learning project

management comes from taking a role in real projects that failed due to insufficient management [5]. By analyzing past situations and evaluating a different path that the project could have taken if specific decisions were made in particular points, a student can enhance his management skills and ability to make decisions. This is similar to the *case study approach* that was developed early in the 20th century and is currently applied in organization planning and administration [6].

However, most of our current project managers are developers who were promoted considering technical skills, without proper training to assume their new responsibilities. Even those who receive some training usually learn through traditional educational strategies, which are content-centric: they focus on “what to learn” instead of “why to learn”. The instructor decides what, when and how learning will be conducted, usually by using classes, textbooks and tests [15].

Two characteristics of software project management present difficulties to the application of the content-centric educational approach. First, it should be noted that only adults undertake project management: so, project management training is adult training. Second, large-scale software projects are complex elements and their behaviour is often too complex for mental analysis.

Concerning adult training, pedagogical studies [9] have shown that the content-centric approach is not adequate for adult learning, since adults prefer to learn based on experience and learn better when they can apply to solve their current problems. Thus, learning by experience and motivation are key issues for better management education.

Concerning complexity, the traditional education approach may not be adequate since project management strongly depends on past experiences and knowledge. While analyzing a decision, a manager usually seeks in his memory for a similar situation in other projects or uses his perception to capture current reality and mentally predict its future state according to available alternatives [4]. This approach requires that the manager has experienced similar situations in the past.

In the learn-by-error approach to management training (implicitly taken when no formal management education is provided to novice managers), this experience comes from participating in failing projects. In the case-study approach, this experience comes from creating and analysing descriptive models of software projects.

However, large software projects are characterized by dynamic complexity in the form of feedback loops, delays, and cause-and-effect relationships distant in time. Their behaviour cannot be efficiently predicted by mental models [17]. Such interpretation often leads project managers to wrong decisions. Since project behaviour cannot be easily derived from basic principles (the content to be learned), the content-centric approach must be

complemented by mechanisms that support experiential learning.

Mentoring novice managers through pilot-projects is an example of such mechanisms. However, it is rarely possible to create real projects for manager trainees due to practical constraints on schedule, budget, and risk.

Another alternative could be the adoption of simulation models of software development projects. Simulation can reduce training time, budget and risks. While real projects can last for months and their failures may have a high cost, students can simulate a similar project model in a few hours, focusing their attention on relevant events occurring throughout the project execution and hiding the details that may confuse the trainee while learning a major lesson. Simulation models can be quickly analyzed and configured for several distinct development situations that could only happen in large projects, with long schedules and large teams.

The use of simulation to support project management education has been analyzed by several studies [10][12]. In a recent paper [14], Pfahl et al. present a controlled experiment to evaluate the effectiveness of using a simulation model in education. In this study, subjects were separated in two groups. One group managed a software project with the aid provided by a simulation model. The second group acted as a control group, using the COCOMO model as a predictive tool for project planning while the experimental group used a simulation model. The results of the study indicate that the use of simulation models provides a better understanding about typical behaviour patterns of software development projects. However, the unique use of simulation models is insufficient to project management education. Simulation is usually a predictive approach: models try to capture some specific real world issues so as simulation can present good insights about the results obtained from particular decisions made. Results are mostly represented by numbers or graphics that are abstract representations of what is really happening within the model during the simulation.

Recently, we have conducted two experimental studies that illustrate some simulation drawbacks for educational goals [2]. Both studies consisted of evaluating the use of simulation to support decision-making on software project management. Subjects were students from two different universities (3 D.Sc. students, 26 M.Sc. students, 16 B.Sc. students and 4 B.Sc.). They were asked to manage a small project with a major objective of concluding it in the lowest schedule as possible, while attending to specific quality restrictions.

A project emulator (that is, a software that dictates project behavior overtime) was used to represent the proposed project. Subjects interacted with the emulator, making decision about which developers should take part in the project team, how many hours should each

developer work per day, if inspections should be included in the development process, and which developer should accomplish each project activity. Half of the subjects used System Dynamics models [7] and simulation to analyze their options and evaluate their decisions before applying them in the project emulator. The remaining subjects managed the project based on their own experience, without the aid provided by the simulator.

The results of these studies show us positive correlation between subject experience, interpretation difficulties and success in attending to project objectives. This was an unexpected result because modeling and simulation were supposed to provide more help to inexperienced managers. Another important issue observed relates to subject engagement. The lack of engagement had negative influence over the subjects' performance. To make modeling and simulation more useful for inexperienced managers, we shall look for better ways to present simulation results. It is usually difficult for a model analyst to trace model observed results to intermediate behaviour.

A problem with simulation tools is their lack of a real project development environment look-and-feel. Since the interaction with the project environment does not resemble a real situation, student's motivation can be limited while using simulation tools. An experiential learning process for project management requires an environment where students can act as managers. Besides, in an artificial learning situation, student motivation and engagement play an important role. Some special drivers for such motivation include self-realization, challenge, victory, rewards, pleasure, and fun. In this sense, games can be integrated to simulation models, adding fantasy, visual effects, and a more compelling interaction model for students. Digital games are also a growing market to adults: the average American player age is 29 years while the average task-force age is 39 years [15]. However, playing is usually considered to be the opposite of working.

Some current research works present the adoption of game concepts in software engineering education, such as the SimSE Tool [13] and the SESAM Project [5]. However, the effectiveness of simulation and game-based learning is a discussion point. Since the educational effects of different approaches are difficult to isolate, measure and trace, their effectiveness is not well documented and established. There are also many disturbing factors that must be taken into account in a comparison, including subjective factors such as the quality of a teacher or a book. Some approaches may be more suitable according to some specific situations and educational goals [8].

3. The Incredible Manager

To evaluate the game-based learning approach, we have developed a simulation game, called *The Incredible Manager*. The diversity of game styles makes it difficult to establish a game taxonomy, but we consider adventure, puzzle, and simulation-games well suited for educational goals aiming at reasoning, judgment, decision-making and system thinking.

By using the game, a student is asked to act as a manager, planning and controlling software projects with success, i.e. within the planned schedule and budget estimates. The game construction is based on three main elements, as can be seen in Figure 1: a simulation model, a simulation machine, and a game machine, which will be detailed on the following sections.

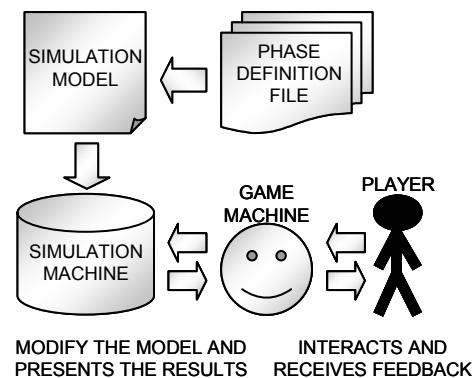


Figure 1. The game structure

4. The Simulation Model

The simulation model represents the world and the aspects that will be simulated and presented to the player. However, software development projects are difficult to model since they are classified as systems of complex dynamics [17].

Addressing these difficulties, System Dynamics [7] is a modeling discipline based on a holistic view to describe and evaluate the visible behaviour presented by a system. Such behaviour is determined by the structure of the elements that participate in the system and the relationships among them. Such structure and relationships are described in the model through mathematical equations. This modeling discipline has already been used in the development of software project models [1], which became a base for subsequent reviews and extensions by other authors.

One of these extensions is the scenario-based project management paradigm [2], which separates uncertain aspects from known facts in project models. This separation occurs by building distinct models (namely scenario models) for each uncertain aspect that can

influence a software project. The models can be more easily developed, modified, integrated and expanded to embrace management knowledge from the technical literature and practice.

Scenario models provide a library of generic management events and theories that an instructor can integrate to a project model and present to management trainees during a simulation session. By using simulations, it is possible to evaluate the impacts of the desired scenarios over the expected project behaviour.

5. The Simulation Machine

The simulation machine is the element responsible for controlling simulation steps, iteratively calculating model equations to evaluate system elements' behaviour. Different from ordinary simulators, the simulation machine for a game must be interactive. Using ordinary simulators, a student playing the role of a manager should prepare a plan (configuring model elements and relationships) and follow it until the end of the simulation. This static structured simulation does not represent with confidence the reality: during a software development project, the manager makes decision all the time during the development process – not only during the planning phase –, modifying the original plan (and thus, the model structure) to better control the project.

The simulation machine developed in our work is able to translate and simulate System Dynamics models and to process events during simulation. This dynamic structured simulation can take into account player actions over the model structure during the game run without rebuilding the behaviour generated by previous simulation steps.

6. The Game Machine

The game machine is the element that the player interacts with and receives visual feedback from the model simulation. It is able to deal with continuous game phases. Each phase represents a separate simulation model, configured externally in a game configuration file. This flexibility allows the adoption of several different educational goals using the same game. The player starts the subsequent phase immediately after finishing the preceding one, even if the later was concluded without success.

During a phase, the project development takes place with hired developers executing a net of project tasks (defined in the model). The characters who take place in the game are:

- Manager - The player's role, responsible for project planning and several decision-making;

- Developers - The team to develop the project. Each one has different skills and characteristics such as hourly cost and work hours per day;
- Boss - Represents all the project stakeholders and is responsible for the project plan acceptance and project pressure during development.

Each game phase is also divided into five steps: Begin Phase, Project Planning, Planning Acceptance, Project Execution and End Phase.

6.1. Begin Phase

The beginning of a phase presents the project to be managed by the player. The project description document includes the description product to be delivered, special scenarios that may impact the development and project characteristics: tasks and its function points, quality, schedule, budget demands and constraints.

6.2. Project Planning

In this step, the player is asked to develop a project plan to be executed. The player must select and hire appropriate developers from those available in the market. Once the team is defined, developers must be assigned to execute the task network. Each task must be executed by only one developer and the player must determine the effort (number of days) necessary to complete each task. The effort on quality assurance activities, such as inspections, is also up to the player decision: the player can even remove these activities from the project plan.

The player can modify the project plan at any time during project execution, firing and hiring developers, modifying their work-hours or modifying the estimated duration of tasks. The project plan resume shows the overall budget and duration estimates to the project.

6.3. Planning Acceptance

Once the project plan is ready, it must be send to the stakeholders for acceptance. The plan can be approved or not. A project plan is refused if its overall estimates are over the constraints described in the project presentation at the begin phase. If the project plan is refused, the player must plan for it again until it is accepted.

6.4. Project Execution

The network task of the accepted project plan is then executed by the allocated team. Figure 2 illustrates the office room where development takes place. The time and funds available for development, as shown in the bottom on the screen, are the ones requested in the accepted project plan. Project execution runs in continuous turns, consuming project resources. The player must be aware of

the project behaviour and take corrective actions when necessary. Visual effects and project reports show the game characters (and model elements) state, such as exhausted developers, late tasks, project without funds, and so on.

To avoid finishing the resources before project completion, the player may need to modify the original plan on the fly. According to these decisions, different players can live the experience of managing the same project in different ways.

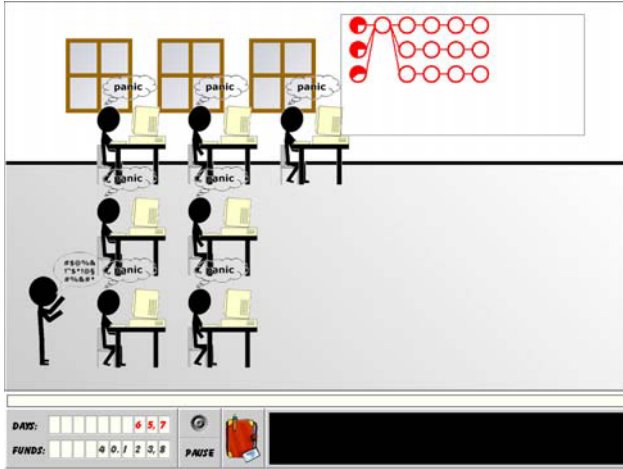


Figure 2. The office room

6.5. End Phase

The phase ends when the project resources are done without project completion (failure) or when all the tasks are done and the project is completed with success.

7. Game-based Learning Evaluation

In the software project management context, to depict the game utility and the improvements that our research should develop, two runs of a case study were conducted to evaluate the adoption of *The Incredible Manager* game within a training concept. The training was divided into simulation and discussion sessions, as stated in [11]. However, only one session of simulation and discussion was applied.

During the simulation session, the subjects were asked to play one phase of the game. When the simulations were finished, an instructor and all subjects participated in a discussion session. The instructor presented common scenarios and approaches of project management, positive and negative examples for specific decision-making situations, allowing the students to better interpret their actions and performance during the simulation session.

The first study was conducted with 7 subjects (1 D.Sc. student and 6 M.Sc. students) from a software project

management course of a Brazilian university. The second study was conducted with 8 subjects (6 M.Sc. students and 2 B.Sc. students) from a laboratory for industrial software development within a different university in Brazil. All subjects received training in project management topics (e.g. function point estimates) and in the game utilization. The training session last 20 minutes in both runs. The first simulation session last between 50 and 120 minutes for the first run and between 55 and 140 minutes for the second run. Although the overall project function points have been kept, the number of project tasks was reduced from 20 to 15, in order to reduce the second study duration, however, without success.

Only one of the subjects reached the end of the game with success (in the first study). Despite the failure, the subject's feedback was considered positive. The training concept with the simulation-game instrument was considered motivating, dynamic, practical and enjoyable. Subjects pointed out some important aspects such as psychological pressures (from continuous-time turns and compelling visual effects), high difficulty as a motivating challenge to the player and the entertainment factor while executing the game without losing the engagement to achieve the goals. Tables 1 resumes the results of the two runs.

Table 1. Game-based Learning Evaluation Results

	<i>Raised</i>	<i>Indifferent</i>	<i>Reduced</i>	
<i>PM Skill</i>	100%	0%	0%	
<i>Interest in PM</i>	87%	13%	0%	
	<i>Good</i>	<i>Indifferent</i>	<i>Bad</i>	
<i>Game-based Training</i>	100%	0%	0%	
	<i>All</i>	<i>None</i>	<i>Lots</i>	<i>Few</i>
<i>Presented Lessons Learned</i>	0%	0%	100%	0%
	<i>Yes</i>	<i>No</i>	<i>Much</i>	<i>Little</i>
<i>Was the training fun ?</i>	47%	0%	53%	0%

The main limitations and drawbacks reported by subjects were related to simplifications that were made to allow the creation of a simulation model for software projects. For instance, our current model is unable to represent real-world situations, such as multiple developers working together in a single task, social interactions among developers, psychological and organizational issues. Some subjects demanded the adoption of a multi-user interface, distance-learning facilities in the game, and some kind of wizard to trace and explain the actions, consequences, lessons learned, and alternative routes for decision-making during the execution of the game. Such wizard would help users to evaluate their own performance after executing the game.

8. Final Considerations

In this paper we analyzed the adoption of practical mechanisms to complement the traditional content-centric education strategies. The current focus is on training software project managers, since the lack of knowledge of management techniques and the inadequate use of management techniques is considered to be a root factor that inhibits project success.

Simulation-based games seem well suited to be introduced in an experiential learning situation, such as required by manager trainees. They give to the student the opportunity of experimenting the consequences of executing or neglecting important project management functions, confront himself with complex issues that must be resolved during project development, and test different approaches and solutions of project management, learning by observing their consequences.

With the evolution of project simulation models, many limitations of current simulation-based game will be addressed to provide a more realistic situation, increasing the number of training scenarios and enhancing knowledge transference. The difficulties of formal models development open a special demand for graphical tools increasing the abstraction level to real world concepts, turning the development of complex models more intuitive and flexible.

To keep up with software project models evolution, the simulation machine presented in this paper should be extended to show more state transitions and graphical feedback, enriching the player perception and entertainment. The simulation machine is able to deal with different models developed upon an existing project management meta-model.

Besides the simulation model, many other research areas can be highlighted: pedagogical evolutions to the training concept with games, art evolutions over game usability and multimedia presentation, research on traces over player actions and performance, and psychological researches about cognitive and motivational issues related to game-based education.

Acknowledgements

The authors would like to thank all the subjects involved in the studies and CAPES for the financial investment in this work.

References

- [1] ABDEL-HAMID, T.K., MADNICK, S.E., Software Project Dynamics - An Integrated Approach, Prentice-Hall, Englewood Cliffs, 1991.
- [2] BARROS, M.O.; WERNER, C.M.L.; TRAVASSOS, G.H., Supporting Risk Analysis on Software Projects, In: The Journal of Systems and Software, v. 71, n. 1-2, pp. 21-35, 2004.
- [3] BROWN, N., "Industrial-Strength Management Strategies", IEEE Software, v. 13, n. 4 (julho), pp. 94-103, 1996.
- [4] DOYLE, J.K., FORD, D.N., RADZICKI, M.J., TRESS, W.S., Mental Models of Dynamic Systems [online]. Available at <http://www.wpi.edu/Academics/Depts/SSPS/Faculty/Papers/27.pdf>. [01/07/2003].
- [5] DRAPPA, A., LUDEWIG, J., Simulation in Software Engineering Training. In: Proceedings of the International Conference on Software Engineering, pp. 199-208, Limerick, Ireland, June, 2000.
- [6] FORRESTER, J.W., System Dynamics and the Lessons of 35 Years, [online]. Available at <http://sysdyn.clexchange.org/sdep/papers/D-4224-4.pdf>, 1991. [05/02/2004].
- [7] FORRESTER, J.W., Industrial Dynamics, Cambridge, MA: The MIT Press, 1961.
- [8] GRÖBLER, A., NOTZON, I., SHEHZAD, A., Constructing an Interactive Learning Environment (ILE) to Conduct Evaluation Experiments. In: Proceedings of the 1999 Conference of the International System Dynamics Society, Wellington, New Zealand, July 1999.
- [9] KNOWLES, M., Andragogy in Action, Jossey-Bass, San Francisco, CA, 1984.
- [10] MAIER, F.H., STROHECKER, J., "Do Management Flight Simulators Really Enhance Decision Effectiveness". In: Proceedings of the 1996 Conference of the International System Dynamics Society, Cambridge, USA, July, 1996.
- [11] MANDL-STRIEGNITZ, P.; LICHTER, H., A Case Study on Project Management in Industry: Experiences and Conclusions. In: Proceedings of the European Software Measurement Conference (FESMA 98), pp. 305-313, Antwerp, Belgium, May, 1998.
- [12] MERRIL, D., Training Software Development Project Managers with a Software Project Simulator [online], Master Thesis Proposal Arizona State University, Tempe, AZ, USA. Available at <http://www.eas.asu.edu/~sdm>, 1995. [01/07/2003].
- [13] OH, E., VAN DER HOEK, A., "Towards Game-Based Simulation as a Method of Teaching Software Engineering". In: Proceedings of the 2002 Frontiers in Education Conference, Boston, MA, USA, November, 2002.
- [14] PFAHL, D., LAITENBERGER, O., DORSCH, J., RUHE, G., "An Externally Replicated Experiment for Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education", Empirical Software Engineering, v. 8, pp. 367-395, 2003.
- [15] PRENSKY, M., Digital Game-Based Learning, McGraw-Hill, 2001.
- [16] STANDISH GROUP, The Chaos Chronicles. The Standish Group International, 2003.
- [17] STERMAN, J.D., System Dynamics Modeling for Project Management. MIT System Dynamics Group, Cambridge, MA, USA, 1992.

A UML-based Software Engineering Methodology for Agent Factory

Rem Collier
University College Dublin
Ireland
rem.collier@ucd.ie

Gregory O'Hare
University College Dublin
Ireland
gregory.ohare@ucd.ie

Colm Rooney
University College Dublin
Ireland
colm.rooney@ucd.ie

Abstract

This paper presents the Agent Factory Development Methodology, an Agent-Oriented Software Engineering (AOSE) methodology that employs a synthesis of the Unified Modelling Language (UML) and Agent UML to support the development of multi-agent systems. We illustrate the use of this methodology, through a simple case study and briefly compare it to some other well-known AOSE methodologies.

1 Introduction

With the continuing emergence of the Agent-Oriented paradigm, there is an urgent need to understand how we might best support developers assigned the task of building an agent-oriented application. This need must be addressed from two perspectives: (1) through the creation of software engineering artefacts (methodologies, tools, architectures etc.) that support the development and deployment of these applications, and (2) through the construction of exemplar applications that act as case-studies illustrating best practices for the use of these artefacts.

The work presented in this paper is concerned with the former of these perspectives, namely the creation of software engineering artefacts that support the development and deployment of agent-oriented applications. Specifically, we introduce the a cohesive methodology that supports the design, implementation, and deployment of agent-oriented applications using *Agent Factory* (AF) [3].

Figure 1, presents a schematic of the AF framework, a four-layer framework that combine: a purpose-built agent programming language known as AF-APL, a distributed FIPA-compliant [5] Run-Time Environment, an integrated Development Environment that supports the implementation and debugging of agents written in AF-APL, and finally, a software engineering methodology that defines a structured approach to the use of the lower layers. This framework is implemented in Java.

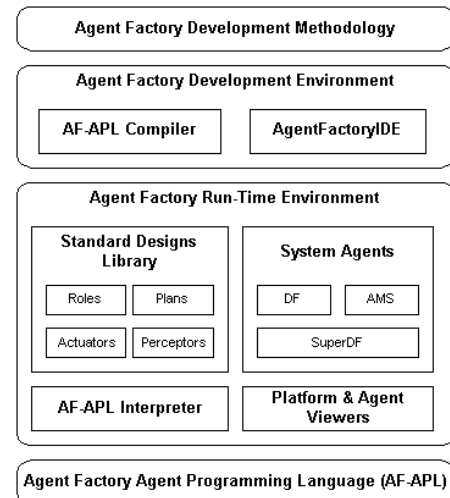


Figure 1. The Agent Factory Framework

To this end, section 2 describes the AF Development Methodology, section 3 highlights the use of this methodology through an exemplar case study of an agent-based internet chat system, and finally, section 4 introduces some related work and presents some concluding remarks. For a more detailed treatment of AF see [3] [4] [11].

2 The Development Methodology

One of the primary objectives behind AF is the development of a cohesive software engineering methodology that delivers structured support for the design, implementation, and deployment of multi-agent systems. In designing this methodology, we sought to address a number of objectives: (1) to employ, where possible, pre-existing industry recognised design notations; (2) to focus upon the definition of visual notations; (3) to use models that promote design reuse; and (4) to maintain a strong link between design and implementation, opening the way for automated code generation. We describe the resultant methodology below.

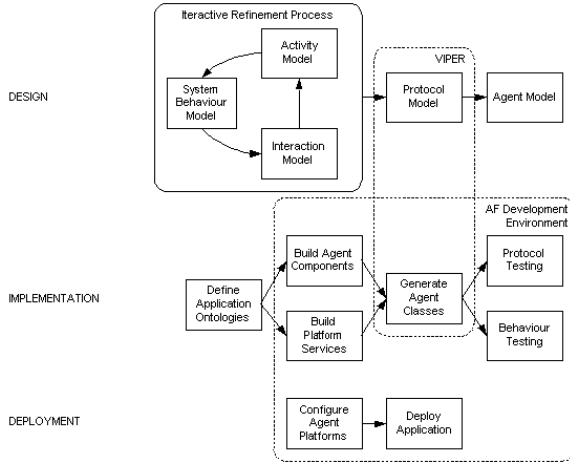


Figure 2. The AF Development Methodology

2.1 The Design Phase

The design phase is concerned with the translation of system requirements into a well defined model of the target system that can be easily implemented using agent technologies. Specifically, we require this methodology to adhere to a number of objectives as set down earlier in this paper. In response to these objectives, we have built the AF Development Methodology around five key models that employ a combination of Unified Modelling Language (UML) and Agent UML [1] diagrams. Figure 2 presents a diagrammatic overview of this methodology.

Central to these models is the notion of an *role*. Within the methodology, roles are used to abstract from fixed agent entities towards the discrete patterns of activity and interaction that will occur in the system. Our rationale for this is that we expect different agents to play the same role during the lifetime of the system. Furthermore, we are able to gain a degree of decoupling between agent implementations and the associated interaction patterns. This allows us to encode best practices for agent interactions as design patterns that may be implemented in a number of ways. This is in line the Foundation for Intelligent Physical Agents (FIPA) efforts at the standardisation of agent interactions (e.g. the Contract Net protocol).

2.1.1 The System Behaviour Model

The first model we develop is the *System Behaviour Model* (SBM). A system behaviour is any distinct set of activities and / or interactions that take place during the operation of the system. For example, in mobile computing system, key system behaviours may include user movement updates, service registration, and service activation.

The principle activity associated with the SBM is the identification of both the key system behaviours and the types of role that the agents will play while engaged in them. In particular, our model distinguishes between two types of system behaviour: (1) *interaction-oriented behaviours* are system behaviours that are associated with two or more roles; and (2) *activity-oriented behaviours* are system behaviours that are associated with a single role.

The SBM is formalised using a customised form of the UML Use Case Diagram. Use Case Diagrams are a well-understood approach for modelling how external actors interact with a software system. From an agent-oriented perspective, we adopt the view of actors as agents that are playing a specific role, and use cases as the behaviours that one associates with these roles. Formally, we customise the UML Use Case Diagram by employing two stereotypes: the `<<role>>` stereotype, which identifies actors that represent roles that will be played by agents, and the `<<role-use-case>>` stereotype, which identifies use cases that occur between agents that are engaged in specific roles.

Upon completion of an initial SBM, the various use cases are organised by behaviour type. These behaviours are then analysed further through the *Interaction Model* (section 2.1.2) and the *Activity Model* (section 2.1.3).

2.1.2 The Interaction Model

The Interaction Model (IM) expands on the SBM through the analysis of the interactions that will occur within each of the interaction-oriented system behaviour. Specifically, for each use case, we identify a number of *interaction scenarios*, each of which describes a potential set of interactions that may take place. Typically, each use case is with one standard scenario (i.e. the one in which everything works out well), together with a number of alternate scenarios that describe variations on the standard scenario. For each scenario, we identify the types of message that the agents send to one another, while playing a role, and the order in which the agents send them.

Given the objectives to employ, where possible, visual design notations, we use a customisation of UML Collaboration Diagrams, similar to that described in [9], to represent individual interaction scenarios. Specifically, we introduce two stereotypes: the `<<role>>` stereotype, which identifies the objects that represent the agents that are playing specific roles; and the `<<fipa-acl>>` stereotype, which constrains the valid message types to the FIPA ACL performatives specified in the FIPA 2000 standards [5].

The purpose of this model is to support the expansion of the initial SBM. As such, it is possible (and in fact expected) that this stage of the process will highlight deficiencies in the initial SBM. Consequently, we expect that the initial IM will be iteratively refined as the analysis progresses. How-

ever, it is expected this model will ultimately reach a level of stability, at which point, it is transformed into a set of protocols within the *Protocol Model* (section 2.1.4).

2.1.3 The Activity Model

The Interaction Model (section 2.1.2) facilitates the expansion of the scenarios that underpin interaction-oriented system behaviours. While this type of behaviour is predominant within agent-oriented applications, it is not the only type of behaviour, there are also activity-oriented system behaviours. To cater activity-oriented behaviours, a third model, known as the *Activity Model* (ActM), is introduced. In contrast with the IM, this model focuses on the activities that underpin the system behaviours.

The ActM is underpinned by the concept of an *activity scenario*. In contrast with interaction scenarios, activity scenarios focus on the activities that the various agents perform while playing associated roles. Initially, this type of scenario was devised for single agent behaviours. However, it has also proved valuable when analysing certain activity intensive multi-agent behaviours. Consequently, the ActM contains at least one activity scenario for each activity-oriented system behaviour, and zero or more activity scenarios for each interaction-oriented system behaviour. In addition, each system behaviour may be associated with multiple activity scenarios. This is because there may be a number of ways that a given behaviour can be realised.

We formalise the ActM through the customisation of UML Activity Diagrams in a fashion similar to that presented in [9]. Specifically, we customise this diagram through the introduction of a <<role>> stereotype, which associates swimlanes with roles.

Again, it is expected that, during the formation of the ActM, various deficiencies in the current design will be identified. As a result, the model will be iteratively refined as the analysis progresses. However, once the model reaches a suitable level of stability, work on the ActM ceases, and the final model becomes an input into the Agent Model (section 2.1.5).

2.1.4 The Protocol Model

Once the first three design models have reached a suitable level of stability, the design process switches from identifying roles, interactions, and activities to their formalisation through two models: the *Protocol Model* (PM) and the *Agent Model*. This section describes the former model.

The PM represents a formalisation of the IM (section 2.1.2). Specifically, the PM refines the various interaction scenarios into a set of protocols that describe how the agents will interact, and which encapsulates each of the alternate scenarios associated with a given system behaviour. Each

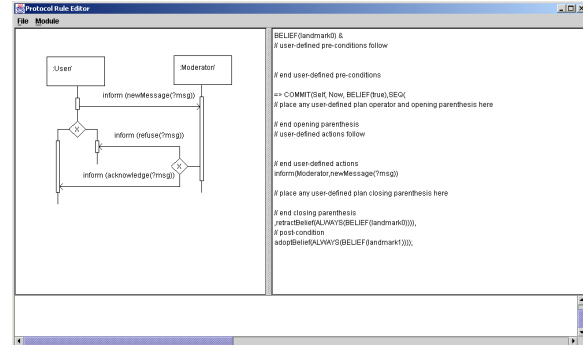


Figure 3. VIPER & Send Message Protocol

protocol specified within this model is defined using Agent UML Sequence Diagrams [1].

It is expected that each interaction-oriented system behaviour will map onto exactly one primary sequence diagram. However, this diagram may itself map onto a number of secondary sequence diagrams that arise from a number of additional refinement activities that can (optionally) take place during the formulation of the model. These activities include: (1) identification and extraction of common interactions within the protocols, the reformulation of these common interactions as *template protocols*; and (2) re-factoring of protocols to make use of any appropriate template protocols, including both those extracted through the previous activity and any known agent design patterns (e.g. Contract Net, Dutch Auction, etc.).

One side effect of this re-factoring is that some re-factored protocols may be linked to system behaviours that have associated activity scenarios. It is vital that the designer be aware of such dependencies, and must re-factor these scenarios to reflect the updated protocols.

Tool-based support for protocol creation is provided via the VIPER [11] visual protocol editor (see figure 3). VIPER performs two jobs within this methodology: (1) it supports visual editing of Agent UML Sequence Diagrams, and (2) it automatically generates agent code based upon these diagrams.

2.1.5 The Agent Model

The Agent Model (AgtM) switches the focus of the design process from the role, interactions, and activities that are necessary to deliver system behaviours to the agents that will exist in the deployed system. Specifically, this model moves the design from a behaviour-centric view to an agent-centric view of the system.

Within this alternate view of the system, we focus upon two concepts: roles, and *agent classes*. Specifically, agent classes represent the types of agents that will be deployed

in the final system, and consequently how the various roles will be *implemented*. We allow a many-to-many correspondence between roles and agent classes. That is, we allow each agent class to be associated with many roles, and each role to be associated with many classes.

To achieve this change of view, we perform three steps: (1) we list each of the roles specified in the SBM, and for each role, list the associated protocols; (2) we associate each role with one or more *agent classes*; and finally, (3) we relate each agent class to the associated set of activities specified within the Activity Model.

Perhaps the most subjective aspect of the AgtM is the selection of which activities to associate with a given agent class. As stated in section 2.1.3, a system behaviour can be associated with a number of activity scenarios. This one-to-many relationship occurs in recognition of the possibility that a system behaviour can be realized in a number of ways (two obvious alternatives are direct realisation versus delegation). The designer must choose which of the potential activity scenarios a given agent class should employ when realising the corresponding system behaviour.

This final model is formalised using a UML Class Diagram that has been customized to include: a `<<role>>` stereotype and an `<<agent-class>>` stereotype. The `<<role>>` stereotype represents roles, and takes the form of a box that contains two compartments: the first compartment contains the stereotype followed by the role identifier, and the second compartment contains a list of protocol identifiers. Conversely, the `<<agent-class>>` stereotype represents agent classes, and takes the form of a box that contains three compartments: the first compartment contains the stereotype followed by the agent class identifier, the second compartment contains a list of protocols (not these specified in the associated roles), and the third compartment contains a list of activity identifiers.

2.2 The Implementation Phase

The second phase of the AF Development Methodology (see figure 2) is the implementation phase. This phase contrasts significantly with the earlier design phase, in that it is tied closely to a specific agent development framework, while the design phase is not.

Central to the implementation phase is the fabrication of a set of agent classes. These classes are implemented in AF-APL [3], an Agent-Oriented Programming language in which agents are mental entities that are modelled using mental attitudes, in this case: *beliefs* and *commitments*. Beliefs describe, using a first-order logic representation language, the current state of the agent and its environment, and commitments describe the current (and future) activities that the agent has decided to perform. Finally, decisions are modelled through a set of commitment rules that map situ-

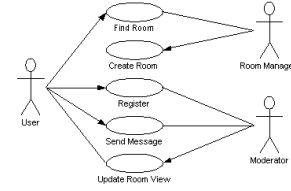


Figure 4. ChatterBot System Behaviour Model

ations (a conjunction of positive and negative beliefs) onto commitments. These rules are checked repeatedly within a sense-deliberate-act cycle.

The beliefs of an agent form an internal model of both itself and its environment. Agent interaction can often involve an agent communicating information held within these beliefs to another agent. To ensure that these internal models are coherent, and to facilitate information dissemination between agents, the process commences with the definition of any application-specific *ontologies*. Within AF, an ontology is realised as a well defined mapping of logical predicates to target domain relations. For example, a mobile computing ontology may specify that the predicate position(?lat, ?long) represents a users position in latitude and longitude.

Upon completion of an initial set of ontologies, the next activity to be carried out is the generation of any custom *agent components* required by the final system. The principle (but not only) agent components employed within AF are perceptor and actuator units. Perceptor units are Java classes that encapsulate specific sensing abilities (e.g. monitor the users location), and which, convert raw data into beliefs. Conversely, actuator units are Java classes that encapsulate specific primitive abilities that an agent may directly execute (e.g. update user profile). The developer identifies potential perceptor and actuator units by reviewing the activities specified in the Activity Model.

In tandem with the construction of agent components, the developer also builds any custom *platform services* required by the final system. A platform service is a service that is deployed on a agent platform as specified by the FIPA-standards [5]. Typical platform services include: message transport, migration, and persistence services.

Once the agent components and platform services have been constructed, the final development activity is started, namely, the generation of AF-APL code that implements the agent classes specified in the Agent Model. In particular, AF realises agent classes as text files, entitled role files, that uses a "rle" extension, and which contain AF-APL code. Reuse of AF-APL code is supported through a USE.ROLE construct similar to the #include construct of C. When converting our design into AF-APL code, we assume a one-to-

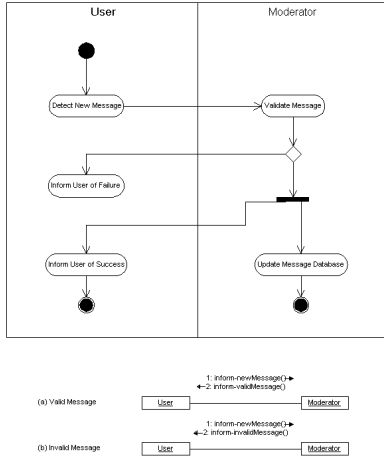


Figure 5. Activity and Interaction Scenarios

one mapping from agent-classes to role files (i.e. role files do not map onto roles as specified in the design, although ongoing work aims to redress this imbalance). Each role file identifies a set of actuators and perceptors that correspond to the specified activities, and a number of commitment rules that describe how those activities should be used.

Finally, the implementation phase concludes with the testing of the implementation. Two types of testing are undertaken: protocol tests evaluate the correctness of the agent interaction protocols, and behaviour tests evaluate the correctness of specific agent behaviours (i.e. that in a given situation, the agent does the expected set of tasks). Once testing is complete, the implementation phase draws to a close, and the system is deployed.

3 Case Study

To illustrate our development methodology, we now present the ChatterBot case study. ChatterBot is an agent-based internet chat system, the basic premise of which is that users are represented by an interface agent that registers the user with a particular chat room, sends messages to that room on behalf of the user, and views any messages posted to the room by other users. When analyzing this system, three key roles were identified: the User role, which is responsible for sending and viewing messages, the Moderator role, that is responsible for validating messages associated with a given room, and the Room Manager role, which is responsible for creating new rooms and managing existing rooms. We formalise this analysis within the System Behaviour Model (SBM) as illustrated in figure 4.

The second phase involves the expansion of each system behaviour specified in the SBM first within context of the

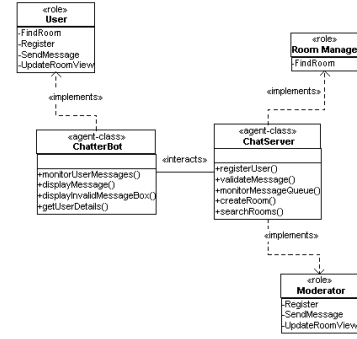


Figure 6. The ChatterBot Agent Model

Interaction Model, and then later within the context of the Activity Model. the of the interaction to build collaboration diagrams for each of the tasks that describe the interactions that should take place between the participants.

By way of illustration, we expand upon the Send Message system behaviour. Our initial expansion of this behaviour is through the Interaction Model. Specifically, we define two interaction scenarios for this behaviour: namely where the User role sends a message that the Moderator role considers to be valid, and where the User role sends a message that the Moderator role considers to be invalid. Figure 5 contains a visual representation of these scenarios.

To help to clarify how the two interaction scenarios arise, we further expand upon the Send Message system behaviour through the Activity Model. Figure 5 presents activity scenario that results from this second expansion. Informally, this scenario indicates that the User role detects a new message has been entered, informs the Moderator role of the new message. The Moderator then validates the message and informs the User role of the success or failure of the validation activity.

As the System Behaviour, Interaction, and Activity models become more stable (i.e. as a general agreement on how the system behaviour emerges), work on these three models stops, and the designer commences work on the Protocol Model. Specifically, the PM is a formalisation of the IM, in which the various interaction scenarios are collapsed into a single protocol. This process is illustrated in figure 3, which presents the Send Message protocol. This protocol is based on the interaction scenarios described in figure 5.

Upon completion of the PM, the final step of the design process involves the formulation of the Agent Model. This final model switches from a behaviour-oriented view of the system to an agent-oriented view of the system. Specifically, this model presents a view of the roles within the system, and the set of agent classes that will implement those roles (section 2.1.5). Figure 6 presents the agent model for ChatterBox. As can be seen in this model, the three roles are

implemented via two agent classes: the ChatterBox class implements the User role, and the ChatServer class implements the Moderator and Room Manager roles.

At this point, the design of ChatterBox is complete and all that remains is for the developers to implement the system. As indicated in section 2.2, this involves the design of a number of agent components and the implementation of the ChatterBox and ChatServer agent classes within AF-APL. Further details of this process are not described here.

4 Discussion and Conclusion

The methodology presented in this paper represents one of a number of potential approaches to fabricating multi-agent systems. Gaia [12] is another methodology, which supports the analysis and design phases. Gaia differs from our methodology in that it is primarily form-based (i.e. not a visual methodology), employs a non-standard design notation, and does not provide any support for the implementation of the designs it produces. In contrast, the MESSAGE methodology [2] does employ a visual design notation, and there is tool-based support for the implementation of multi-agent system. However, while MESSAGE employs the same meta-modelling language as UML, MESSAGE diagrams bear little resemblance to UML diagrams, which are well understood within the software industry. A third methodology is that outlined in [6]. Like our methodology, Heinze's methodology does employ UML use cases and activity diagrams, and does specify the link between design and implementation. However, Heinze's methodology does not account for multi-agent interactions, nor is there any tool-based support for the development process.

In summary, this paper presents a visual agent-oriented software engineering methodology that is founded upon the industry standard UML design notation. Furthermore, the models underpinning this methodology facilitate both design reuse, and automated partial code generation. Specifically, partial tool-based support for the methodology currently exists, in the form of VIPER, a visual protocol editor [11]. Finally, due to space constraints, this paper has illustrated our methodology through a trivial case study. However, this methodology has been employed in the development a number of large scale agent-based applications including the WAY System [7], Gulliver's Genie [10] and the award-winning ACCESS Architecture [8].

References

- [1] B. Bauer, J. P. Muller, and J. Odell. Agent uml: A formalism for specifying multiagent interaction. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*. Springer Verlag, 2001.
- [2] G. Caire, W. Coulier, F. J. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. E. Kearney, J. Stark, R. Evans, and P. Massonet. Agent oriented analysis using message/UML. In *AOSE*, pages 119–135, 2001.
- [3] R. W. Collier. *Agent Factory: A Framework for the Engineering of Agent-Oriented Applications*. PhD thesis, Department of Computer Science, University College Dublin, 2001.
- [4] R. W. Collier, G. M. P. O'Hare, T. D. Lowen, and C. F. B. Rooney. Beyond prototyping in the factory of agents. In *Proceedings of 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003)*, Prague, Czech Republic, 2003.
- [5] FIPA. Fipa 2000 standards. URL: <http://www.fipa.org>.
- [6] C. Heinze, M. Papasimeon, and S. Goss. Specifying agent behaviour with use cases. In *Proceedings of the Pacific Rim International Workshop on Multi-Agent Systems (PRIMA 2000)*, pages 128–142, 2000.
- [7] T. Lowen, G. O'Hare, and P. O'Hare. Mobile agents point the way: Context sensitive service delivery through mobile lightweight agents. In C. Castelfranchi and W. Johnson, editors, *Proceedings of First International Joint Conference on Autonomous Agents and Multi-Agent Systems Conference (AAMAS 2002)*, Bologna, Italy, 2002. AAAI Publishers.
- [8] C. Muldoon, G. OHare, D. Phelan, R. Strahan, and R. Collier. Access: An agent architecture for ubiquitous service delivery. In M. Klusch, A. Omicini, S. Ossowski, and H. Laamanen, editors, *Cooperative Information Agents VII*, LNAI 2782. Springer Verlag, 2003.
- [9] J. Odell, H. V. D. Parunack, and B. Bauer. Extending uml for agents. In *Proceedings of AOIS Workshop at AAAI 2000*, 2000.
- [10] G. O'Hare and M. O'Grady. Gulliver's genie: A multi-agent system for ubiquitous and intelligent content delivery. *Computer Communications*, 26(11):1178–1187, 2003.
- [11] C. F. B. Rooney, R. W. Collier, and G. M. P. O'Hare. Viper: Visual protocol editor. In *Proceedings of COORDINATION 2004*, Pisa, Italy, 2004.
- [12] M. Wooldridge, N. R. Jennings, and D. Kinny. The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

ADAMS: an Artefact-based Process Support System

Andrea De Lucia, Fausto Fasano, Rita Francese, Genoveffa Tortora

Dipartimento di Matematica e Informatica

Università degli Studi di Salerno

Via Ponte don Melillo, 84084, Fisciano (SA), Italy

adelucia@unisa.it, ffasano@unisa.it, francese@unisa.it, tortora@unisa.it

Abstract. *We present ADAMS (ADvanced Artefact Management System), a Web-based system that integrates project management features such as resource allocation and process control and artefact management features, such as coordination of cooperative workers and artefact versioning, as well as context-awareness and artefact traceability. Maintaining traceability links (dependencies) between artefacts supports management of changes during incremental and iterative software development in a flexible way. Basically, the traceability layer is used to propagate events concerning changes to an artefact to the dependent artefacts, thus also increasing the context awareness in the project.*

1. Introduction

In the last decade a lot of research effort has been devoted to the development of methodologies and technologies supporting coordination and collaboration of distributed software engineering teams. Examples are Computer Supported Cooperative Work (CSCW) and groupware, workflow management, and configuration management. Configuration Management (CM) is among the others mostly used in software engineering projects to face with coordination problems. CM tools (see e.g., [5, 11, 18, 21]) help to coordinate the activities of developers, by providing capabilities that either avoid parallel development altogether (e.g., locking) or assist in resolving conflicts (e.g., merging). Independently of the adopted model (Checkout/Checkin, Composition, Long Transaction, ChangeSet), existing CM systems are based on the *workspace* concept, representing the work environment of each user [19]. The adoption of such separate areas causes a lack of *context-awareness*, as a developer is informed of work made by others on the artefacts he/she is working on or on related artefacts, only after these have been checked-in, thereby significantly delaying the discovery of potential problems.

Process Support Systems (PSSs) [3, 9, 16], including Workflow Management Systems (WfMSs) [12, 13, 22] and Process-centered Software Engineering Environments

(PSEEs) [1, 4, 14], represent a different research area aiming at supporting the coordination of software development activities through process modelling and enactment. Despite the advances made in the field, most of the solution proposed have not gained wide acceptance in the industry because the Process Description Languages (PDLs) they propose for the modeling of business processes are too complicated to understand and manipulate. Most of them are activity-based and model software processes in a top down manner, focusing on the specification of the control and data flow between activities. In these systems the modeling of the activities is often similar to programming, it is difficult and laborious and, as a consequence, they do not facilitate the work of the project manager. Most of them do not support the deviations from the process model when unforeseen situations (frequently) happen and even when such a support is provided, it is too complicated to manage these situations within the process support system. Also, the production of an artefact is seen as the result of the execution of an activity and often there is lack of integration with configuration management systems [3]. With respect to CM tools, most recent PSSs provide a grater support to context-awareness, by integrating communication tools and notification mechanisms to make aware developers about events occurring within activities [3, 9, 14, 16].

Both PSSs and CM tools generally do not offer an adequate support to artefact traceability and, as a consequence, handling changes is difficult. CM tools mainly enable versioning of artefacts, but traceability information among different artefacts is lacking and when supported, the traceability infrastructure fails during the system evolution [8]. In PSSs dependencies between artefacts can be derived from the data flow links between activities but relationships between the artefacts produced during the software development process is not directly stored and maintained.

In this paper we present ADAMS (ADvanced Artefact Management System), a Web-based system that integrates project management features such as resource allocation

and process control and artefact management features, such as coordination of cooperative workers and artefact versioning, as well as context-awareness. Rather than defining the control and data flow between activities, like in most PSSs, software processes in ADAMS are modelled through the produced artefacts and the relations between them. Maintaining traceability links (dependencies) between artefacts supports management of changes during incremental and iterative software development in a flexible way. Basically, the traceability layer is used to propagate events concerning changes to an artefact to the dependent artefacts, thus also increasing the context awareness in the project.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents an overview of ADAMS, while Sections 4-6 describes features provided by the system related to artefact management, cooperation and context awareness, and traceability, respectively. Finally, Section 7 discusses concluding remarks and future work.

2. Related Work

Several Configuration Management systems are available, some of which are open source. BitKeeper [5] is a scalable CM system supporting distributed software development. CVS [11] is one of the most used open source versioning tools: it is based on a central repository and provides support for branching and merging. Perforce [18] is a scalable and lightweight tool supporting the concept of change and change set. StarTeam [21] is an innovative solution produced by Borland for handling Web software development.

Palantir [19] is a system that complements Configuration Management tools by providing support to the context awareness. It offers a graphical display to the developer for verifying which remote artefacts are changing and give an evaluation of the severity and of the impact of changes.

Recent PSSs also address problems related to the integration of process modelling and enactment with artefact management and context-awareness problems [3, 9, 15]. GENESIS is an open source PSS supporting software engineering processes in a highly distributed environment [3]. The process modeling language is activity-based and enables the decomposition of complex processes into sub-processes that can be distributed and executed at different organizational sites. GENESIS integrates an artefact management subsystem, namely OSCAR [17], designed to non-invasively interoperate with work-flow management systems, development tools, and existing repository systems. Similarly to ADAMS, artefacts in OSCAR have a type hierarchy, similar to the object-oriented style. Every artefact possesses a collection of standard meta-data, and is represented by an XML

document containing both the meta-data and the artefact data.

PROSYT, like ADAMS, adopts an *artefact-based* approach [9]. Each artefact produced during the process is an instance of some *artefact type*, which describes its internal structure and behaviour. PROSYT also allows for distributed enactment facilitated by an event-based middleware [10]. It is able to tolerate deviations from the process model during enactment.

Maurer proposes a tool named MASE (MILOS for Agile Software Engineering) that enables virtual software teams to adopt distributed extreme programming [15]. It is based on the MILOS system [16], that supports the dynamic coordination of distributed software development teams over the Internet. In particular, MILOS is a web based WfMS allowing for dynamic changes on the project plan during project enactment and provides a notification mechanism which tracks product changes and informs involved people.

The Ophelia project aims at developing a platform supporting software engineering in a distributed environment [20]. Among the others, it offers a traceability layer that enables traceability across all project artefacts. In Ophelia artefacts of the software engineering process are represented by CORBA objects. A graph is created to maintain relationships among these elements and can be used for navigating between them. ADAMS also offers such browsing facility. In addition, ADAMS enables developers to subscribe events on artefacts of interests, in order to receive notifications about changes to these artefacts.

Chen and Chou [7] have proposed a method for consistency management in the Aper process environment. The method is based on maintaining traceability relations between artefacts and using triggers to identify artefacts affected by changes to a related artefact.

Cleland-Huang *et al.* [8] have developed EBT (Event Based Traceability), a traceability method based upon event-notification. Software artefacts are linked by a publish-subscribe relationship. When a change occurs on a given artefact having the publish role, notifications are sent to all the subscriber (dependent) artefacts.

The latter two papers are very closed to the approach developed in ADAMS. However, unlike ADAMS, they do not offer facilities to developers to directly subscribe/unsubscribe events. Indeed, the main problems of EBT, are the higher number of messages that are generated within a process if too many links are maintained [8] and the fact that often traceability links are not correctly maintained during a software development process [2]. ADAMS gives developers the possibility of customizing the set of events they would like to be notified about, thus avoiding undesired notifications on one hand and receiving notifications that are not planned by the

traceability layer on the other hand. Another distinctive feature of ADAMS with respect to these two related approaches is the support for the management of the entire life cycle of artefacts that includes a checklist-based inspection and review phase.

3. ADAMS overview

ADAMS (ADvanced Artefact Management System) is an artefact-based process support system. It enables the definition of a process in terms of the artefacts to be produced and the relations among them, supporting a more agile software process management than activity-based PSSs, in particular concerning the deviations from the process model.

ADAMS poses a greater emphasis to the artefact life cycle by associating software engineers to the different operations that can be performed on an artefact. In particular, ADAMS provides support for the definition of artefact types with related standard templates and for a checklist-based inspection and review phase of the artefact life cycle. Software engineers are given the possibility to subscribe for particular events concerning artefacts and projects, thus increasing the context-awareness level.

ADAMS provides functionality to manage resources, projects, and artefacts. In particular, the system enables the definition of roles within a project. Standard roles are:

- Administrator, who manages the system itself, including artefact types and human resources; he/she also defines projects and the corresponding project managers and allocate resources to them;
- Project Manager, who manages the resources allocated on a project, defines the artefacts to be developed and the corresponding artefact managers, allocate resources to artefacts, and define artefact dependencies;
- Artefact Manager, who manages the evolution of an artefact and defines roles and permissions for software engineers working on it.

ADAMS has a web-based architecture. The system is decomposed into six subsystems with a layered architecture (see Figure 1). These modules are independent, so that it is possible to change one of them, without affecting the global system integrity. The presentation layer is implemented in HTML and JSP. The application logic layer is composed of four subsystems, namely the Artefact Management Subsystem (AMS), the Project Management Subsystem (PMS), the Administration Subsystem (AS), and the Event Management Subsystem (EMS). The first three subsystems are implemented as Java Servlets and use the EMS that is responsible for managing subscriptions and notifications of events. In particular, the AMS manages

artefact types, the lifecycle of the artefacts, and the traceability layer, while the PMS provides functionalities for project creation, resource allocation, calendar and scheduling activities. Access to the database is achieved through the functionalities offered by the persistent data access subsystem.

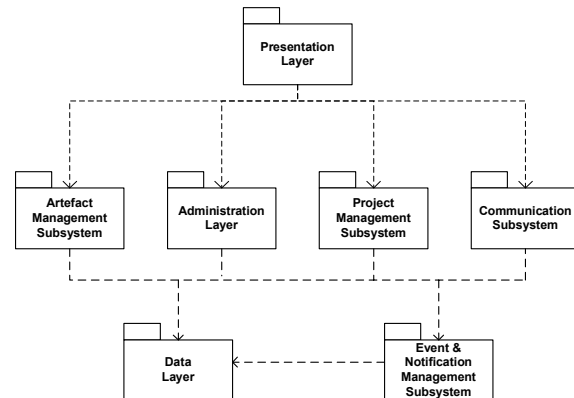


Figure 1. ADAMS architecture

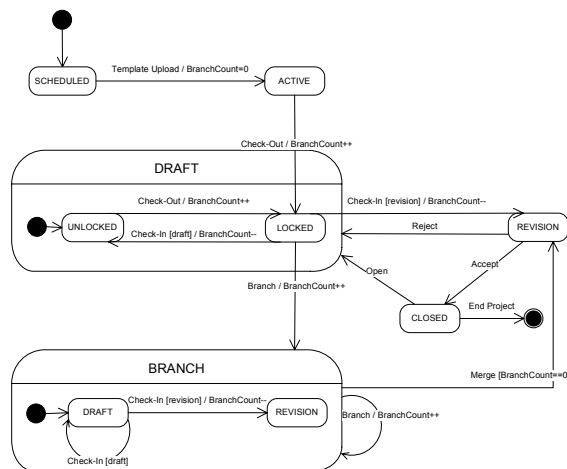
4. Artefact Management in ADAMS

Artefacts play a central role in ADAMS. The administrator can define artefact types according to the standards included in the quality manual of the subject organization. Artefact types can be associated to standard templates as defined in the quality manual. These can be customized during the definition of the quality plans of specific projects and as starting point for the development of artefact type instances. Besides a standard template, a checklist can be associated to an artefact type and used during the review phase of the artefacts of that type.

Artefacts can be either simple files or can be a hierarchy of simpler artefacts. For example, a software requirement specification document includes several functional and non-functional requirements; each requirement can be considered a simpler artefact that might affect different parts of the software architecture and evolve independently of other artefacts of the same type. To this aim hierarchies of artefact types can be defined in a recursive way starting from file types. A Document Type Definition (DTD) is automatically created and associated to each artefact type and used to check that artefacts of that type respect the artefact type definition.

Artefacts in ADAMS follow the general life cycle depicted in Figure 2. This, together with the resource permissions definition and management, represents a first process support level and allows the Project Manager to focus on the practical problems involved in the process and avoid to get lost in the complexity of the process

The higher level artefacts to be produced within a project are defined (scheduled) by the project manager, who also associates the needed resources to work on them and choose the artefact managers. The artefact manager uploads the template and checklist defined in the project quality plan, defines permissions of resources to work on the artefact and activates it. The artefact manager can also schedule the sub-artefacts the subject artefact is composed of and associate resources to them. Also, he/she can give permissions to define sub-artefacts to other resources, although he has the duty of linking the sub-artefacts to the higher level artefact.



Once activated, several draft versions of an artefact can be created and maintained by ADAMS. When scheduled, the manager can decide if branches are allowed during the production of an artefact. If branches are not allowed, each resource can lock the artefact (check-out) and work on it until a new version is uploaded and checked-in. Otherwise, different branches can be produced and worked independently by each resource (see Figure 3), who can also produce different versions of each branch. When all branches are closed, they can be merged in a new version of the artefact. Completed non-branch versions of an artefact undergoes the revision process and are either approved and closed or sent back to the draft state.

In hierarchical artefacts, leaves are associated with files (simple artefacts) and follows the life cycle described above, while internal nodes define composite artefacts. Therefore, operations that can be performed on internal

Artefact No.17 Card	
Name	SOURCE
Status	BRANCH
Creation Date	23/01/03
Start Date	01/02/03
End Date	30/06/03
Manager	Fausto Fasano
Project	ADAMS
Artefact Type	SOURCE
Last Version	2.0
Branch Allowed	<input checked="" type="checkbox"/>
Branches	2
Concurrent Users	Fausto Fasano Andrea De Lucia

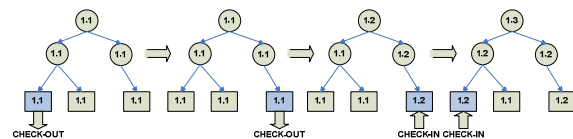


Figure 4. Versioning of hierarchical artefacts

The support for cooperation is offered by ADAMS through typical features of a configuration management system. ADAMS enables groups of people to work on the same artefact, depending on the required roles. Software engineers can cooperate according to a lock-based policy or concurrently, if branch versions of the artefact are allowed. At any time it is possible to see the people who are working on an artefact (see Figure 3). As discussed in the previous section, ADAMS also provides support for software inspection.

Besides these functionalities, the system has been enriched with features to deal with some of the most common problems faced by cooperative environments, in

particular context awareness and communication among software engineers.

Context awareness is mainly supported through event notifications: software engineers working on an artefact are notified when another branch is created by another worker. This provides a solution to the isolation problem for resources working on the same artefact in different workspaces: in fact context awareness allows to identify possible conflicts before they occur, since the system is able to notify interested resources, as soon as an artefact is checked-out and potentially before substantial modifications are applied to it.

ADAMS also enables software engineers to subscribe events they would like to be notified about. Events mainly concern the operations performed on artefacts and projects. For example, an event could be the modification of the status of an artefact or the creation of a newer version for it. A number of events are automatically notified without any need for subscription. Examples include notifying a software engineer he/she has been allocated to a project or an artefact.

Events concerning the production of new versions of an artefact are also propagated through the traceability layer of ADAMS to the artefacts (and consequently to their managers) depending directly or indirectly on it (see Section 6).

ADAMS provides direct communication mechanisms between software engineers, such as e-mails. Moreover, cooperation between software engineers during iterative software processes is supported through the possibility of sending feedbacks concerning software artefacts. Software engineers that make use of previously developed artefacts to produce new artefacts might send feedbacks to the input artefacts in case of problems (see Figure 5). Feedbacks are then notified to artefact managers to make decisions about. Feedbacks and event-based traceability are the two mechanisms used by ADAMS to support process management.

6. Support for traceability

Besides providing versioning and composition of artefacts, ADAMS provides support for artefact traceability. The project and artefact managers can create and store traceability links between artefacts either in the same hierarchy (e.g., between two functional requirements in the requirement document) or in different hierarchies (e.g., between a functional requirement and a module in the design document).

The traceability links between artefacts involved in the same project are modelled in terms of dependences between them. A dependence consists of a relation between two artefacts, together with some additional information to specify the type of dependence, as starting conditions, production constraints and output rules.

Besides being useful for impact analysis during software evolution, traceability links in ADAMS are also useful to manage the software process and notify software engineers that the production of a given artefact can start, or that an artefact has to be changed, because of some changes in artefacts it depends on. Dependencies can be mandatory or optional for the software development process. Through the dependencies it is possible to specify several things of the development process, such as whether the production of an artefact needs a previously developed artefact in draft or complete form and if the new artefact will be an update of a previous artefact or the latter has to be considered only as an input for the production of the new artefact.

This event-based traceability approach to process management, in addition to feedbacks, is much more flexible than activity-based workflow management systems, in particular with respect to the deviations from the process model. In addition to event propagation through the dependencies, the traceability links can be visualized by a software engineer, while showing the artefact he/she is working on (see Figure 5), and browsed to look at the state of previously developed artefacts and download latest versions, or to subscribe events on them, in order to receive notifications concerning their development.

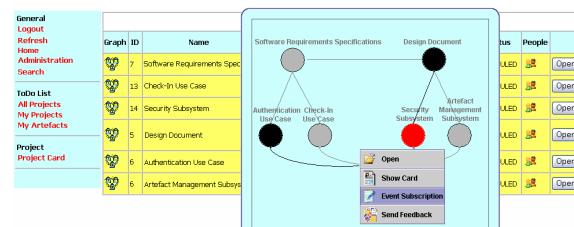


Figure 5. Traceability visualization and event subscription

7. Conclusion and future work

In this paper we have presented ADAMS, an artefact based process support system that integrates project management and artefact management features. Besides coordinating distributed software engineers working on the same artefact and providing versioning facilities like configuration management systems, ADAMS also support context-awareness through event subscription and notification and artefact traceability. Traceability links are used to propagate events concerning changes to an artefact to the dependent artefacts, thus also increasing the context awareness in the project.

ADAMS has been implemented as a web-based system using Java technologies, Apache Tomcat 4.1 as web

server and MySQL 4.0 as Database Management System. The user interface of the system is implemented by 63 Java Server Pages. The code for the application logic and data layer is composed of about 35K lines of java code, spread among 20 servlets implementing the application logic subsystems and 65 java beans providing data layer functionalities. The database is composed of 32 database tables.

ADAMS is currently being experimented in the Software Engineering course of the Computer Science program at the University of Salerno (Italy). Experimentation includes about 60 students involved in seven different projects.

Future work includes the integration and experimentation in ADAMS of information retrieval techniques to support a technical manager in correctly identifying and maintaining traceability links between evolving software artefacts [2]. Indeed, one of problems we have noticed from the preliminary results of the experimentation of ADAMS is the fact that often software engineers fail in identifying and maintaining traceability links between software artefacts in incremental and iterative processes. Another advantage of using these techniques in ADAMS is the fact that they can provide feedbacks concerning the consistent use of domain terms within related software documents.

References

1. V. Ambriola, R. Conradi, and A. Fuggetta, "Assessing Process-Centered Software Engineering Environments", *ACM Transaction on Software Engineering and Methodology*, vol. 6, no. 3, 1998, pp. 283-328.
2. G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation", *IEEE Transaction on Software Engineering*, vol. 28, no. 10, 2002, pp. 970-983.
3. L. Aversano, A. De Lucia, M. Gaeta, and P. Ritrovato, "GENESIS: a Flexible and Distributed Environment for Cooperative Software Engineering", *Proceedings of 15th International Conference on Software on Software Engineering and Knowledge Engineering*, S. Francisco, California, USA, 2003, pp. 497-502.
4. S. Bandinelli, E. Di Nitto, and A. Fuggetta, "Supporting Cooperation in the SPADE-1 Environment", *IEEE Transactions on Software Engineering*, vol. 22, no. 12, 1996, pp. 841-865.
5. BitKeeper Home Page. <http://www.bitkeeper.com>.
6. F. Casati, S. Castano, M. Fugini, I. Mirbel, and B. Pernici, "Using Patterns to Design Rules in Workflows", *IEEE Transactions on Software Engineering*, vol. 26, no. 8, 2000, pp. 760-784.
7. J.Y.J. Chen and S.-C. Chou, "Consistency Management in a Process Environment", *The Journal of Systems and Software*, vol. 47, 1999, pp. 105-110.
8. J. Cleland-Huang, C. K. Chang, and M. Christensen, Event-Based Traceability for Managing Evolutionary Change, *IEEE Transaction on software Engineering*, vol. 29, no. 9, 2003, pp. 796-810.
9. G. Cugola, "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models", *IEEE Transactions on Software Engineering*, vol. 24 no. 11, 1998, pp. 982-1001.
10. G. Cugola, E. Di Nitto, and A. Fuggetta, "The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS", *IEEE Transactions on Software Engineering*, vol. 27, n. 9, 2001, pp. 827-850.
11. CVS Home Page. <http://www.cvshome.org>.
12. P. Grefen, B. Pernici, and G. Sánchez (Eds.), *Database Support for Workflow Management: The WIDE Project*, Kluwer Academic Publishers, February 1999; ISBN 0-7923-8414-8.
13. D. Georgakopoulos, H. Hornick, and A. Sheth, "An Overview of Workflow Management: from Process Modelling to Workflow Automation Infrastructure", *Distributed and Parallel Databases*, vol. 3, no. 2, 1995, pp. 119-153.
14. J. C. Grundy, M. D. Apperley, J. G. Hosking, and W. B. Mugridge, "A decentralized architecture for software process modeling and enactment", *IEEE Internet Computing*, vol. 2, no. 5, 1998, p 53-62.
15. F. Maurer, "Supporting Distributed Extreme Programming", *Proceedings of 2nd Conference on XP Agile Universe*, Chicago, IL, USA, 2002, Springer Verlag, Lecture Notes in Computer Science Series, vol. 2418 pp. 13-22.
16. F. Maurer, B. Dellen, F. Bendeck, S. Goldmann, H. Holz, B. Kötting, and M. Schaaf, "Merging Project Planning and Web-Enabled Dynamic Workflow for Software Development", *IEEE Internet Computing*, vol. 4, no.3, 2000, pp. 65-74.
17. D. Nutter, S. Rank, and C. Boldyreff, "Architectural requirements for an Open Source Component and Artefact Repository System within GENESIS". In *Proceedings of the Open Source Software Development Workshop*, University of Newcastle (UK), 2002, pp. 176-196.
18. Perforce Home Page. <http://www.perforce.com>.
19. A. Sarma and A. van der Hoek, "Palantir: Coordinating Distributed Workspaces", *Proceedings of the 26th IEEE Annual International Computer Software and Applications Conference*, Oxford, UK, IEEE Computer Society Press, 2002, pp. 1093-1097.
20. M. Smith, D. Weiss, P. Wilcox, and R. Dewer, "The Ophelia traceability layer", in *Cooperative Methods and Tools for Distributed Software Processes*, A. Cimitile, A. De Lucia, and H. Gall (editors), Franco Angeli, 2003, pp. 150-161.
21. StarTeam Home Page. <http://www.starbase.com>.
22. Workflow Management Coalition, "Workflow Management Coalition Interface 1: Process Definition Interchange Process Model", Document no. WFMC-TC-1016-P, 1999, available from <http://www.aiim.org/wfmc/standards/docs/if19910v11.pdf>.

Agent Technology Portfolio Manager

K. S. Barber, J. Ahn, N. Gujral, D. N. Lam and T. Graser

The Laboratory for Intelligent Processes and Systems, Electrical and Computer Engineering,

The University of Texas at Austin, Austin, TX 78712

{barber, jsahn, ngujral, dnlam, graser}@lips.utexas.edu

Abstract. *Software agents are becoming a new means of designing and building complex, distributed software systems. Developing large-scale agent systems requires the proper selection of agent technologies where selection is based on where adherence to the agent architecture structure and satisfaction of domain (functionality, data, timing) and installation requirements. The diversity of agent technologies and the lack of a common framework for describing these technologies challenges designer attempting to evaluate, compare, and potentially reuse agent technology. The technology selection and evaluation process requires an adequate representation of agent technologies in the context of the agent-oriented competencies (e.g. sensing, modeling, planning, acting) it can fulfill, the domain-related functionality it can perform, and its requirements of installation. This paper describes a repository of agent technologies, Technology Portfolio Manager (TPM) toolkit, which will assist the agent designer in evaluating and selecting technologies which map to the desired agent architecture and requirements.*

1. Introduction

Software agents have become an expressive and useful archetype for developing complex software architectures. Software agents encapsulate functionality that enables them to perform activities on their own or through interaction (e.g. coordination). To help software architects manage the complexity of specifying an agent-based architecture and selecting appropriate agent-related technologies, an Agent Competency Framework has been defined [2]. Agent competencies define the minimum capabilities an agent should have, and thus, defined the functional notion of agency. In terms of functionality, an agent is an entity that senses and acts in its environment, models the environment and objects in its environment, and deliberately plans or reacts towards a given goal [1]. Thus, the core agent competencies are sensing, modeling, planning and acting (Figure 1).

Communication, organization and coordination are additional competencies required to conduct some/all of the core competencies within multi-agent systems (as opposed to a single agent system). Agent Competencies (ACs) offer fundamental building blocks for defining and specifying all types of agents [7]. Given an agent architecture which results from Designer's Agent Creation and Analysis Toolkit [7], the architect can use the Technology Portfolio Manager to browse available agent-based technologies that satisfy the competencies included in the architecture.

A multi-agent system (MAS) designer is guided by specific desired capabilities and properties, in the context of a particular domain, for the entire system and/or a particular agent. Thus, agent technologies are developed / selected for a MAS by considering their application to a particular domain and their ability to offer desired capabilities (competencies). Consequently, the designer must have a means for viewing and comparing agent technologies with respect to both competencies provided and domains supported. However, when attempting to compare various agent technologies or simply understand the breadth of agent technologies, the designer encounters obstacles that include the disparity in how agents are modeled and the lack of separation between domain functionalities and domain independent functionalities (Agent Competencies). As a result of this diversity, agent developers have difficulty comparing different views of agent technology or even different implementations of the same agent technology on some common basis [7].

This research effort populates a tool, the Technology Portfolio Manager (Figure 2), with four sets of information and the relationships between these sets: agent competencies, agent infrastructure services, domain tasks in a particular domain (in this case UAV surveillance) and agent technologies. Using the Technology Portfolio Manager, the designer can effectively compare technologies using a common representation for describing agent technologies and an intuitive interface that relates technologies to agent competencies and domain tasks. Furthermore, the tool

allows the designer to interrogate the repository from the perspective of:

- technologies (displaying related domain tasks and competencies satisfied by the selected technologies),
- domain tasks (displaying related competencies and technologies capable of delivering selected domain tasks),
- agent competencies (displaying related domain tasks and technologies capable of satisfying the selected competencies).

The basic definition of the agent competency ontology is described at Section 2. The complete functionality and the scope of the agent technology repository specified in the Technology Portfolio Manager will be discussed in the following sections.

2. Agent Competency Ontology

The Agent competency ontology is the key for describing (1) the critical criteria for agency and (2) correlate domain tasks such as “Generate UAV routing” to domain-independent competencies such as “planning” and, in general, (3) offer a common framework for representing and comparing agent technologies. The Agent Competencies provide a specification of agent capabilities that distinctly delineate agent functionalities. By specifying agent technologies in terms of these agent functionalities, different views of agent design can be functionally compared and a common understanding among agent software engineers is promoted. Agent competencies were based on the essential set of domain-independent functionalities an agent delivers.

As seen in Figure 1, there are two types of agent competencies that form the framework for specifying agents. Core Competencies (CCs) define the essential functionalities of an agent. Pluggable Competencies (PCs) are also defined because agents interact with other agents and entities in the system. PCs are not essential in single-agent systems, but they need to be considered in multi-agent systems [1] [7].

In addition to agent technologies, agent implementations may depend on many infrastructure related technologies necessary for design, run-time operations, and analysis [5][6][8]. The following subsections describe the agent competencies and infrastructure services.

2.1. Sensing

The agent needs to acquire appropriate data from other agents and the environment. The sensing competency is composed of two sub-competencies, data acquisition and data preprocessing.

Data acquisition is the internalization of data that is external to the agent obtained by sensors or other agents. Data preprocessing is the preliminary manipulation of data that can be utilized by the agent. The output of sensing is the sensed data useful for the agent reasoning processes, such as semantic variables and variables packaged into data structures.

2.2. Modeling

Modeling is the maintenance of the information specified by the developer and/or derived from sensed data. This information is critical for the performance of the agent. If the information about the agent’s environment is inaccurate, it may behave in seemingly undesirable ways. An agent may model information about itself, other agents in the system, available resources, the environment, and objects in the environment.

The modeling competency is decomposed into two sub-competencies – variable characterization and model revision. Variable characterization is the association of properties to a variable. Given variables originating from sensor readings or other agents, the agent can associate properties. Model revision is the integration of beliefs from variable characterization into the current model. An agent can collaborate by sharing beliefs and coordinate on verifying the consistency of models.

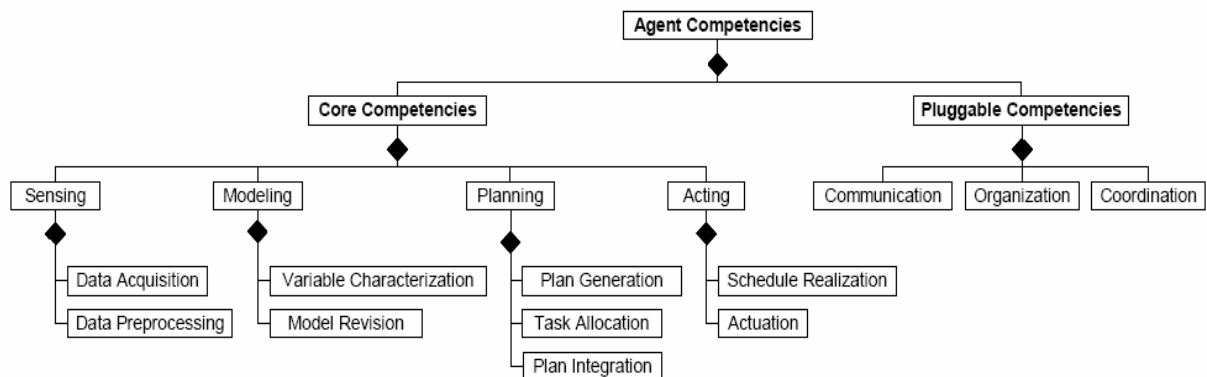


Figure 1: Decomposition of Agent Competencies

2.3. Planning

For each goal that an agent has selected to achieve, the agent must select actions to perform and schedule the execution of those actions in an effective manner as to achieve the goal or bring the agent closer to the goal. In the pursuit of goals, agents need the capability to generate actions that are possible in its current situation, choose the appropriate action(s), and decide when and by whom those actions will be executed. The planning competency is decomposed into three sub-competencies - plan generation, task allocation, and plan integration.

2.4. Acting

Schedules of actions are received and handled by the acting competency of the agent, which filters out actions that are not relevant and executes the appropriate actions at the appropriate times. The acting competency is decomposed into two sub-competencies – schedule realization and actuation.

2.5. Pluggable Competencies

In addition to CCs, when an agent operates in a multi-agent system, it should also have the functionality to communicate, to form organization(s), and to coordinate with other agents in the multi-agent system (MAS). These competencies are not included as core competencies since single-agent systems may not need such functionality. Instead, communication, organization, and coordination are Pluggable Competencies (PC) because they work in conjunction with and in the context of CCs. PCs support the inter-agent functionality that CCs may require in a MAS.

The communication competency is used to interoperate with other agents, which is essentially the intentional transfer of information. An organization imposes societal structure on an otherwise disorderly multi-agent system and establishes the relationships among agents. Since data and control are distributed among the agents, coordination is needed to prevent conflicts or inconsistencies that occur among agent tasks.

2.6. Infrastructure Services

Agent competencies (especially core competencies) are associated with notion of agency. Additionally every agent implementation has an associated infrastructure, thus some of the agent technology providers focus on infrastructure services. Tom Wagner and Omer Rana [3] have pointed out that building multi-agent systems requires a large amount of software infrastructure and many systems require

planning, scheduling, transport, coordination, communication, simulation, and module integration technologies. Star and Ruhleder [4] mentioned that infrastructures have the general character of being embedded inside other structures; transparent (not needing reinvention or re-assembly each time); of wide reach or scope; learned as part of community membership; linked to conventions and norms of community practice; embodying standards, shaped by pre-existing installed bases of practice and technology; and invisible in use yet highly visible upon breakdown.

Therefore, we propose agent infrastructure services categorized by: design, run-time, analysis.

- *design* services are used during the design process to specify multi-agent system (e.g., design methodology),
- *run-time* services are used as part of a multi-agent system execution environment (e.g., security module),
- *analysis* elements are needed for analyzing the behavior of multi-agent systems (e.g., evaluation module).

3. Knowledge Acquisition Process

For this research effort, technologies to be described in the Technology Portfolio Manager (TPM) were developed as part of the Defense Advanced Research Project Agency - Taskable Agent Software Kit program. The TASK program was initiated with the specific intent to advance state-of-the-art agent technology as well as promote tools for easy agent-oriented design and analysis. The process of populating the repository was conducted in several phases beginning with the collection of all the information available about a technology in presentations and papers posted by the technology providers (e.g. researchers) involved in the DARPA TASK program. Following initial modeling efforts, every Technology Provider was interviewed to (i) verify the technology models, specifically the mappings and (ii) obtain additional information which might have been missed during the interpretation phase from the gathered papers and presentations. Mappings are the relationships between the domain-specific capabilities of the technology, and the domain independent agent competencies and infrastructure services.

The models for mapping between the respective technology, agent competencies and the domain-specific capabilities were verified with the technology providers. During the mapping process it was realised that not all technologies were designed to offer agent competencies but instead provided a foundation design, operation or evaluation for the MAS through their services. The notion of *infrastructure services* was introduced in order to map these technologies in the repository tool. Thus, the Knowledge Acquisition

Reports, document the sources of the information as well as the rationale for the mappings between the technologies, CCs, PCs, ISs and the Domain Tasks. The mappings and the rationale were confirmed or verified through individual meetings with the technology providers. After the mapping between the technologies and the competencies was finalized, the information and relationships were populated into the TPM.

4. Technology Portfolio Manager (TPM)

In the context of the Technology Portfolio Manager, a “technology portfolio” is defined as a collection of technologies where the specifications of respective technologies offered by different solution providers can be showcased in order to understand their potential contribution to an MAS design. The TPM is intended to aid a designer when deciding, which technologies to select for an agent design, depending upon the competencies those technologies cover in a particular domain. There are different queries that can be asked depending upon what the user desires. This section will explain the queries that can be issued by the user, the intent or motivation of the user or designer in issuing

that query and then explain the results or outputs from the tool. For this example, the TPM has been loaded with the DARPA TASK Agent Technology Repository, a collection of agent technology specifications acquired and represented by UT-Austin in the DARPA TASK program for the UAV surveillance domain. As seen in Figure 2, the screen displays (1) Agent Competency and infrastructure service ontology; (2) agent technologies from various providers; (3) domains to which agent technologies have been applied; and (4) tasks in a selected domain. The interface also allows users to issue queries by selecting one or more competencies, infrastructure services, technology providers, technologies, domains, or tasks. The different areas of query formation can be seen as tabs on the tool main window (Figure 2). The issued query can be seen at bottom of the screen.

4.1. Queries selecting Agent Competencies and Infrastructure Services

The most common query is: “For one or more agent competencies and/or infrastructure services, what technologies provide the selected competencies or infrastructure services, and which domains and domain

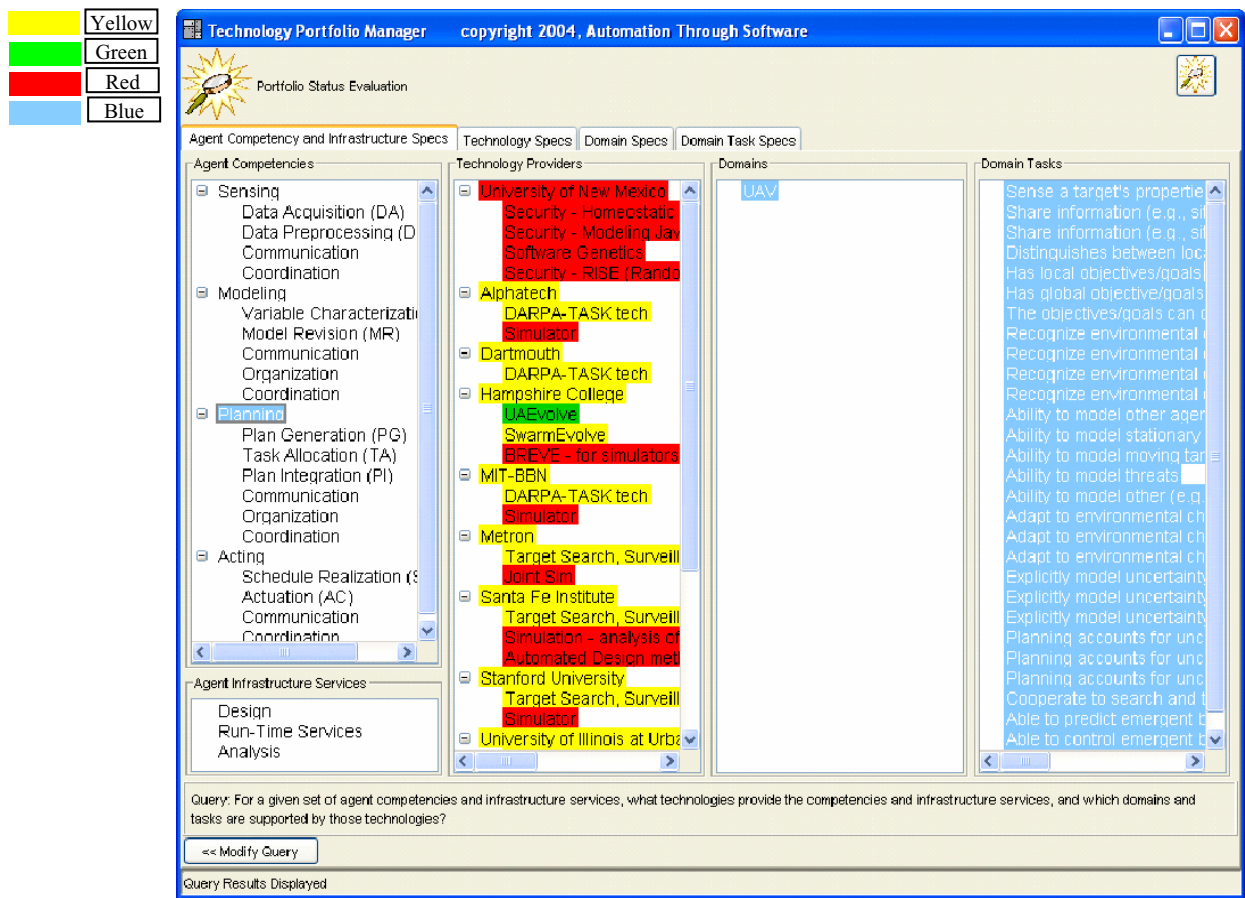


Figure 2: Query with respect to Agent Competencies

tasks are supported by those technologies?” This query is issued by simply selecting (mouse click) one or more agent competencies or infrastructure services in the left most column. The MAS designer may want to know if there are some technologies which are able to provide some or all of the desired competencies or infrastructure services. Additionally, for those technologies shown to deliver the selected competencies or infrastructure services the TPM displays domain and domain tasks the technologies can deliver. For the first part of the query it can be seen that the tool responds by indicating the user-selected core-competency in blue (in this case “Planning”) and colors the related technologies. Technologies (in the second column) in red do not support the competency, while technologies in green and yellow fully and partially support the competency, respectively. The result of the second part is not shown here which shows the highlighting of the domains and domain tasks which are supported by the technologies in green and yellow (Figure 2).

This query helps the designer in comparing the technologies based on desired competencies related to domains and domain tasks. If the designer wants to know in detail if there are some technologies which are able to provide some or all of the desired sub-competencies or infrastructure sub-services and to discover if those technologies support any domains and constituent domain tasks then the query will be: “For the given agent sub-competencies or infrastructure sub-services, what technologies provide given sub-competencies or infrastructure sub-services, and which

domains and tasks are supported by those technologies?” This query is much similar to the first query described but requires the user to select sub-levels of the competencies or infrastructure services. The tool lets the user select multiple competencies or infrastructure services (both core and sub) as the criteria for making a query.

4.2. Queries selecting Technologies

By selecting a technology provider or a respective technology the user issues the following query: “For given technology providers (technologies), what competencies and infrastructure services do technology providers (technologies) support, and to which domains and tasks have those technologies been applied?” The designer for MAS may want to know which competencies and infrastructure services are provided by the selected technology providers and/or technologies. The results of the above query are shown in the Figure 3. For the first part of the query it can be seen that the TPM responds by indicating user-selected technology provider of interest in blue (in this case “Metron”) and coloring the related competencies. Competencies are colored based on the constituent sub-competencies supported by the technology: red for none, yellow for some, and green for all. Domain tasks to which one of the technology provider’s technologies has been applied are colored in green, and the “UAV” domain is colored in yellow, indicating that the technology provider does not have technologies that

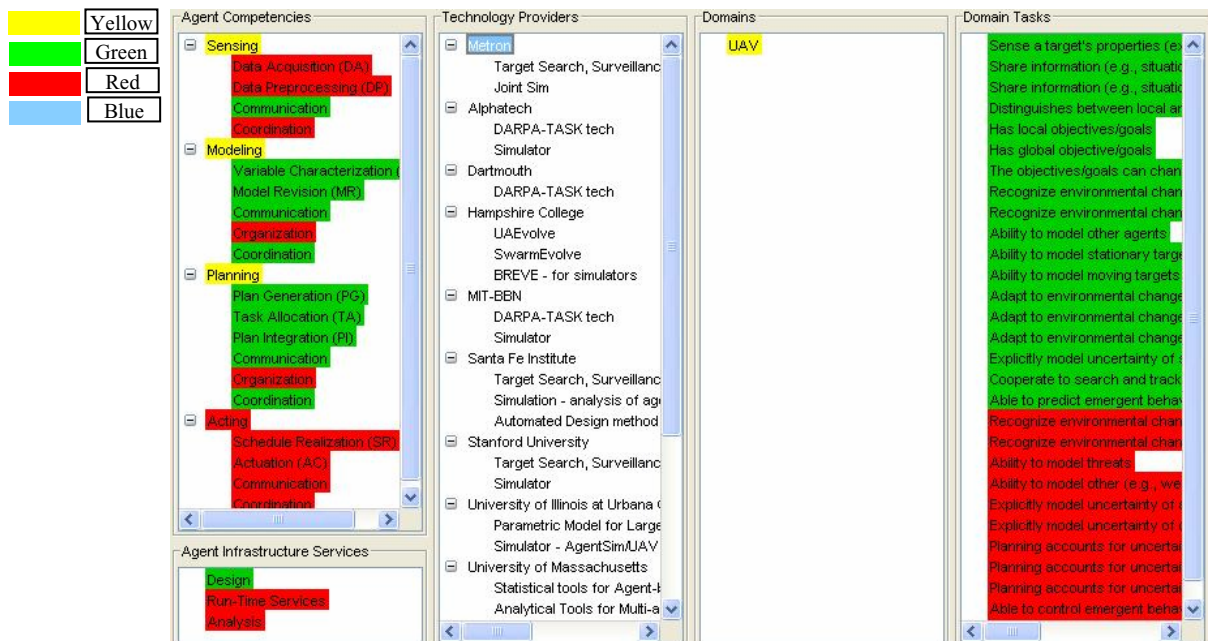


Figure 3: Query with respect to Technologies

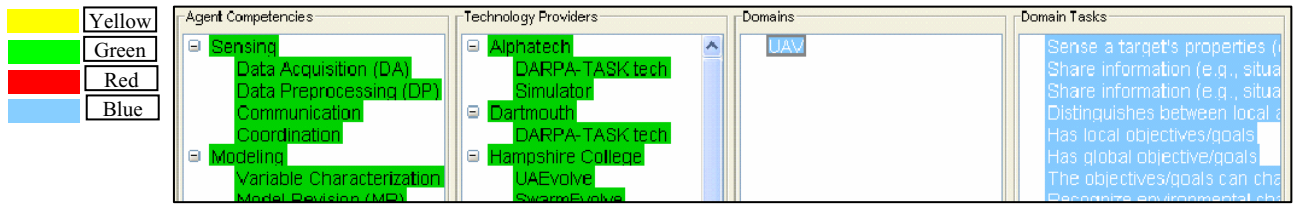


Figure 4: Query with respect to Domains

cover all the tasks defined under the “UAV” domain. This query helps the designer in getting a crystal picture of how much a technology provider covers a particular domain, the agent competencies, and infrastructure services.

The other query under this area of interest could be: “For given technologies, what competencies and infrastructure services do technologies support, and to which domains and tasks have those technologies been applied?” This query is more detailed, targeting one or more technologies. The designer may want to go into more detail to know which are the sub-competencies and infrastructure sub-services provided by some or all of the desired technologies and if those technologies have been applied to respective domains and their domain tasks.

4.3. Queries selecting Domains

The user might also be interested in the domain, thus the query in this case would be: “For the given domain(s), what technologies have been applied to given domain(s), and which competencies and infrastructure services do those technologies support?” The designer may want to know which technologies have been applied to the selected desired domains and if the technologies support any of the competencies and infrastructure services. As shown in Figure 4, the tool responds to the user selecting a domain by highlighting in blue all tasks associated with the “UAV” domain, and coloring green all technologies (and corresponding technology providers) that have been applied to at least one of the “UAV” domain tasks (Column 4). Competencies and sub-competencies are colored green if they are supported by any of the highlighted technologies. This query helps the designer in getting a coherent view of how much a domain has been worked

on by different technology providers and the competencies that can be applied to the domain and its tasks.

4.4. Queries selecting Domain Tasks

By selecting one or more domain tasks, the user asks: “For the given domain tasks, what technologies have been applied to given domain tasks, and which competencies and infrastructure services do those technologies support?” When using this query, the user is asking if any of the technologies have been applied to some or all of the desired domains tasks and if the technologies support some or all of the competencies and infrastructure services. As shown in Figure 5, the tool responds to the user-selected domain tasks by highlighting the corresponding domain (in this case “UAV” domain) and coloring green all technologies (and corresponding technology providers) that have been applied to at least one of the selected domain tasks. Competencies and sub-competencies are colored green if they are supported by any of the highlighted technologies. This query helps the designer in getting a coherent view of how much every or a group of domain tasks has/have been covered by different technology providers and the competencies that are mapped to these domain tasks.

5. Related Work

To date, researchers have developed some fairly sophisticated theories and technologies to be used in agent systems. With the ever-increasing demand for new software and the improvements in multi-agent systems, widespread use of well-designed and well supported agent technology offers tremendous opportunity [10]. However, progress in this area of

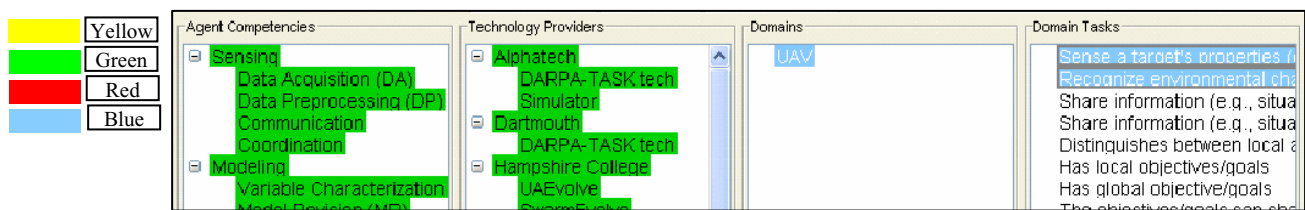


Figure 5: Query with respect to Domain Tasks

agent software design is largely dependent on the creation of better technology component libraries and improved access mechanisms. Numerous design tools for developing agent-based system have been proposed. (e.g., agentTool [11], JAF [12], RETSINA [13], Agentbuilder [14]). But building large scale multi-agent systems from existing technology components still remains a distant reality [9]. The Technology Portfolio Manager (TPM) described herein assists the designer in comparing technologies using a common representation for describing agent technologies and an intuitive interface that relates technologies to agent competencies and domain tasks. Through TPM, the designer can easily access existing agent technologies and select technologies which map to the desired agent architecture and requirements.

6. Summary

When designing a software system architecture using available technology components, an architect evaluates various technology combinations with respect to the degree to which selected technologies meet stated requirements. For a Multi-Agent System architecture, technologies are evaluated with respect to (i) agent-related “competencies” provided (core capabilities that characterize agency including planning, acting, sensing, modeling, communication, organization and coordination), (ii) “infrastructure services” offered (e.g. design, runtime, and analysis services), and (iii) domain tasks supported (i.e., the problem domain being addressed by the agent system). The Technology Profile Manager (TPM) described in this paper provides the designer with a tool for browsing a repository of agent technology specifications. Unique features of the tool include the following: (i) The designer can evaluate how well different technologies map to the desired agent competencies or infrastructure services. (ii) The respective coverage of every technology and its provider can be analyzed with respect to a domain and respective domain tasks.

These features support the designer when performing the types of trade-off and what-if analysis that are associated with selected agent technology and deriving agent-based system designs.

In addition to describing the Technology Portfolio Manager, this paper outlined the procedure followed for building a repository within the TPM modeling agent technologies developed by participants in the DAPRA TASK program in the context of a UAV search and surveillance domain. A significant step in the procedure involved interpreting technology information obtained from technology providers and mapping that information to the “agent competency” and “infrastructure services” ontology defined by researchers at the University of Texas at Austin.

As part of ongoing research, the current repository mappings will be extended to incorporate refinements to the existing “competency” and “infrastructure service” ontology. The TPM tool will also be integrated into a suite of tools being developed to support agent-based architecture definition and analysis.

7. Acknowledgement

This research is sponsored in part by the Defense Advanced Research Project Agency (DARPA) Taskable Agent Software Kit (TASK) program, F30602-00-2-0588. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Project Agency.

8. References

- [1] K. S. Barber and D. Lam, “Specifying and Analyzing Agent Architectures using the Agent Competency Framework,” *15th International Conference in Software Engineering and Knowledge Engineering*, Redwood City, CA, 2003, pp. 232-239.
- [2] K. S. Barber and D. N. Lam, “DACAT: Designer's Agent Creation and Analysis Toolkit,” The University of Texas at Austin, Austin, TX, Technical Report TR2003-UT-LIPS-007, May 20, 2003.
- [3] T. Wagner and O. Rana, Preface in Lecture Notes in Artificial Intelligence: *Infrastructure for Agents, Multi-Agent Systems and Scalable Multi-Agent Systems*, Volume 1887, Springer-Verlag Berlin Heidelberg, January, 2001.
- [4] S. L. Star and K. Ruhleder, “Steps to an Ecology of Infrastructure,” *Information Systems Research*, vol. 7, 1996, pp. 111-138.
- [5] L. Gasser, “MAS Infrastructure: Definitions, Needs and Prospects,” In Lecture Notes in Artificial Intelligence: *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, Volume 1887, Springer-Verlag Berlin Heidelberg, January, 2001, pp. 1-11.
- [6] K. S. Barber, A. Goel, D. C. Han, J. Kim, D. N. Lam, T. H. Liu, M. T. MacMahon, R. M. McKay, and C. E. Martin, “Sensible Agents: An Implemented Multi-Agent System and Testbed,” *The Autonomous Agents and Multi-Agent Systems (AAMAS-2001)*, Montreal, Canada, p. 92-99.
- [7] K. S. Barber, D. N. Lam, “Architecting Agents using Core Competencies,” *Autonomous Agents &*

- Multiagent Systems (AAMAS-2002)*, Bologna, Italy, pp. 90-91.
- [8] Mihhail Matskin, Ole Jorgen Kirkeluten, Sverre Bjarte Krossnes, Oystein Sale "Agora: An Infrastructure for cooperative work support in multi-agent systems," In Lecture Notes in Artificial Intelligence: *Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, Volume 1887, Springer-Verlag Berlin Heidelberg, January, 2001, pp. 28-39.
 - [9] D. Garlan, R. Allen, J. Ockerbloom "Architectural Mismatch or Why it's hard to build systems out of existing parts" Proceedings of *the Seventh International Conference on Software Engineering*, Seattle, Washington, April, 1995, pp. 179-185.
 - [10] K.S. Barber, Sutirtha Bhattacharya "A Representational Framework for Technology Component Reuse", Proceedings of *the 13th International Conference on Software and Systems Engineering and their Applications (ICSSEA 2000)*, Grenoble, France, pp. 285-288.
 - [11] Scott A. DeLoach & Mark Wood, "Developing Multiagent Systems with agentTool." in *Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop, Lecture Notes in Computer Science*, Vol. 1986, Springer Verlag, Berlin, 2001, pp. 46-60.
 - [12] Vincent, Regis; Horling, Bryan; and Lesser, Victor. "An Agent Infrastructure to Build and Evaluate Multi-Agent Systems: The Java Agent Framework and Multi-Agent System Simulator". In *Lecture Notes in Artificial Intelligence: Infrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems*, Volume 1887. January, 2001, pp. 102-127.
 - [13] K. Sycara, J.A. Giampapa, B.K. Langley, and M. Paolucci, "The RETSINA MAS, a Case Study," *Software Engineering for Large-Scale Multi-Agent Systems: Research Issues and Practical Applications*, Alessandro Garcia, Carlos Lucena, Franco Zambonelli, Andrea Omici, Jaelson Castro, ed., Springer-Verlag, Berlin Heidelberg, Vol. LNCS 2603, July, 2003, pp. 232-250.
 - [14] AgentBuilder, Inc, "Agent Construction Tools" <http://www.agentbuilder.com/AgentTools/>, 2/2001

AgentService

Antonio Boccalatte¹ Andrea Gozzi² Alberto Grosso³ Christian Vecchiola⁴

Department of Communication, Computer, and System Sciences

{¹nino, ²gozzi, ³agrosso, ⁴cristian}@dist.unige.it

Abstract

AgentService is an agent-programming framework based on the Common Language Infrastructure (CLI). AgentService exploits CLI's innovative features like the application domains and remoting to design its core elements. In this paper, the modular architecture, the agent model, and the scheduling system of AgentService will be discussed. Particular attention will be given to the issues that make AgentService a productive, effective, and customizable agent-programming environment.

1. Introduction

The need of interoperability with new technologies has made software complex. Software developers no longer have the time to build real and effective applications from scratch, thus a great emphasis is now placed on reusable code, reusable components, and reusable patterns [1]. Today's programmers rely on pieces of code from many different sources that work together correctly and reliably, providing new applications with useful features. Component Software [2], a development methodology in which independent pieces of code are combined to create application programs, has arisen in response to this trend. Component-aware software environments are now widely adopted in professional software development.

Object-oriented programming and Component Software are definitely valuable technologies, but sometimes a different approach is more useful and effective. Dynamic environments where relations among components frequently change are designed and implemented effectively by using the agent-oriented paradigm. Agents' theory was especially designed for these scenarios: agents adapt their behaviour to the mutable conditions of the environment and interact to achieve the best performance. Multi-agent systems (MASs) are the common expression of agent-oriented programming. Building a multi-agent system is difficult: MASs application programming requires skills in traditional distributed and concurrent systems. Furthermore, additional difficul-

ties arise from flexibility requirements and the need of sophisticated interactions.

This paper presents a solution that takes advantage of both agent-oriented programming and Component Software. AgentService is an agent-programming framework based on the Common Language Infrastructure (CLI). The CLI is the programming and execution environment of AgentService: agents developed with AgentService have full access to the wide world of components and can seamlessly exploit the wide set of services offered by means of the CLI. In the following sections, the architectural model and the design decisions made in the implementation of AgentService will be discussed: particular attention will be given to those issues that make AgentService a productive, effective, and customizable agent programming environment.

2. Background

2.1. Agents and Multi-agent Systems

A software agent is an autonomous software entity with some level of "intelligence" [3]. While a complete definition of software agent is still an open issue [4], some qualities that characterize this concept are widely accepted by the community of researchers. Agents are situated in an environment and interact with it in order to meet their design objectives. By means of cooperation, negotiation, and competition agents interact with other agents to enhance their performance. These qualities characterize the agent's social ability. Pro-activity and reactivity make the agents exhibit a flexible and autonomous behaviour. Pro-activity is the ability to show goal directed behaviour by taking initiative, while reactivity is the ability to cope with the environmental changes in a timely fashion. Pro-activity and reactivity define agents as intelligent software entities. For these reasons, agents are more sophisticated structures than objects: they have control over their behaviour (not only over their state); agents interact with a set of messages that define a conversation (not by method invocation). Nevertheless, they are implemented as autonomous, concurrent,

and self-contained objects.

Agents have been designed to operate in a community. Such a community is normally called a multi-agent system (MAS) [5]. MASs are decentralized systems with distributed control and asynchronous computation, they provide a runtime environment and define the infrastructure for the agent interaction and communication. MASs provide a set of useful services to agents as message transport system and directory services. An abstract specification of a generic MAS was proposed by the Foundation of Intelligent Physical Agents (FIPA) [6], an international organization promoting standards for agent technologies. The architectural model proposed by FIPA is commonly used as a reference in comparing different MASs implementations.

2.2. The Common Language Infrastructure

Common Language Infrastructure is an ECMA [7] and ISO-IEC [8] standard that defines a virtual execution environment. CLI is a component oriented programming platform where language-agnostic modules of code are executed in a secure manner.

The Common Language Infrastructure has been designed to serve different programming languages. It offers a full-featured class library and a wide set of runtime services that guarantee proper code execution. Language interoperability is one of the most interesting features of the CLI: modules written in different programming languages can be seamlessly and effortlessly integrated without building ad-hoc software connectors. Although language interoperability can also be implemented with other technologies (i.e.: the java virtual machine), such a possibility has not been exploited while it is possible to write CLI-oriented applications in more than 20 programming languages.

The Common Language Infrastructure comes with two innovative concepts: *application domains* and *remoting*. The application domain is the unit of execution inside the CLI and also defines the granularity of the security policies applied to code execution. Many application domains can be hosted in a single process and each application domain can have different permissions. Application domains are "lightweight address spaces" and CLI enforces restrictions on passing data between domains: objects that wish to communicate across domain boundaries must use special communication channels and behave according to specific rules. This technique, referred to as remoting, can be used to communicate between application domains running on different physical computers, operating systems, and processors.

Different implementations of the CLI made this programming and executing environment available on many operating systems and hardware platforms. A shared source implementation of CLI is SSCLI also known as Rotor [9].

3. The AgentService Programming Platform

AgentService is an agent-programming platform developed on top of the common language infrastructure. The core features of AgentService are:

- a new agent model that simplifies the design and the implementation of agents;
- a multi-agent system that hosts agents and define their runtime environment;
- language support to the development of agents with Agent Programming eXtensions (APX);
- a complete integration with the CLI that allows access of the wide world of components to the agents.

In the following the previously described features will be investigated in detail and a particular attention will be given to the scheduling of the agents and their activities.

3.1. Agent Model

Within AgentService, agents are designed as software entities whose activity is defined by a particular managed set of data (Knowledge objects) and performed by a set of concurrent tasks (Behaviour objects). Knowledge objects hold all the information shared by different behaviour objects and they set and constitute the knowledge base of the agent. Knowledge objects are persisted and maintained by the platform in order to have a fail-safe recovery in case of system crash. In this way, agents are able to maintain their state safely. Behaviour objects define the agent tasks and contain all the agent computational logic. Behaviours have been designed to be concurrent and they do not have to be divided into different steps to allow concurrency (concurrency is guaranteed by the run-time environment of the agents). Behaviours can also have proper private data, but such data are not persisted like the knowledge objects.

The distinction between activities and data allows a clear decomposition of the agent definition. Furthermore, it is important to note that knowledge objects can hold any type of objects defined in the CLI, while behaviours are designed as components so they can fully exploit all the features of the object-oriented model defined in the CLI. Such a flexible structure allows the developers to perform a fast prototyping of agents: new behaviour and knowledge types can be programmed, or they can be chosen from ready-to-use libraries. Hypothetically, the agent designer only needs to know the features of the components and assemble them into an agent definition without writing any line of code for the agent computation. Of course, this is not likely to happen, but the simple reusability of already designed components (i.e. knowledge and behaviour objects) is an important advantage since it speeds up the implementation and

allows the design of safer code (components already used probably contain less bugs than new ones).

While many agent programming frameworks adopt the BDI architecture [10], AgentService offers a more flexible model. The BDI architecture, as well as other agent architectures, can be effortlessly implemented with the proposed model. As pointed out by Rao and Georgeff [11], the implementation of a BDI architecture leads to the definition of the following elements:

- *belief set*;
- *goal set*;
- *plans*.

Beliefs and goals can be easily modelled with knowledge objects, while a proper implementation of plans is given by the behaviour objects. An additional behaviour acting as the BDI engine needs to be designed. Such a behaviour looks at the knowledge representing the goals and schedules the proper behaviour plans according to the information stored in the knowledge representing the belief set.

AgentService distinguishes between the definition of agent and its live instance at run-time. The formers are called agent templates, while the others are the real agents. The relationship between the two entities is similar to the relationship between the class type and its instances: each agent instance acts according to the corresponding agent template. It is worth noting that the instances of agents do not strictly have the type of their template, but complete different classes are instantiated at run-time. By means of reflection, the agent template is inspected and the agent instance is created according to that model. The running instance of the agent has all the necessary stubs to exploit the platform services and to interoperate with other agents, while the agent templates and the behaviour objects deal with proxies of these services only. The advantages of such separation are:

- the behaviour designer is able to program the behaviour components as if they were plugged in the run-time host that is the agent instance;
- the agent definition and the agent instance remain loosely coupled;
- the run-time support can be easily customized also by varying the implementation of those stubs needed by the agent template;
- the run-time support can be also modified in its structure without affecting the agent template interface (in case of inheritance relationship this task is harder to accomplish).

The separation between the agent definition and its running instance offers interesting advantages. Common implementations of the agent model use an abstract agent class specialized by inheritance in order to define new agent types. The types used at runtime are the same defined by the programmers. Such a model leads to high coupling and reduces the possibilities of customization. By using the proposed agent model, different components and hosts can be configured for the AgentTemplate derived classes, so that a more effective support can be obtained. Even if AgentService is built on a portable runtime, specific platform services can be adopted to enhance the performance. The high level of componentization allows the easy integration of these services.

3.2. Platform Architecture

The AgentService platform has been designed following the FIPA abstract architecture specifications [12]. The FIPA guidelines require the presence of some compulsory components that handle the core functionalities of the multi-agent system. These components are implemented as agents and they are:

- the agent management system (AMS): it is the supervisor and the controller of the MAS, it is responsible for the agent scheduling;
- the directory facilitator (DF): it offers directory services (i.e. yellow pages) to the agents of the platform and its external clients;
- the message transport system (MTS): it handles the messaging system of the platform and the inter-platform communication.

By using the multi-behavioural model previously described, the different tasks performed by each component have been modelled with the abstraction of the behaviour. Since these agents represent the core components of the platform, particular attention has been given to their design: while the communication among the agents programmed by developers can occur by the messaging systems only, each agent has a direct access to the core agents. This is a critical issue because agents lose the control over their behaviour by direct method invocation. It is worth noting that agents have access to these elements by using strict interfaces and proxies that prevent agents from having explicit object references to the core agents. In this way AMS, DF, and MTS maintain their autonomy and provide the best performance.

The platform is divided into several modules that can be configured in order to adapt the platform to different contexts; some services might be not critical for the platform activity and can be switched off if they are not required.

Fundamental modules have the same degree of customization of the complementary ones; moreover, ad-hoc implementations can be provided to better exploit the features of the operating system that hosts the platform. These modules cover:

- the messaging subsystem and infrastructure;
- the persistency subsystem;
- the storage management system.

AgentService provides a standard implementation of these components relying only on the Common Language Infrastructure services, yet specific and ad-hoc implementation can be provided during installation time. AgentService deals with them seamlessly by means of clear-cut interfaces: AMS, MTS, and DF are automatically configured with the installed components in a transparent manner. These components along with the core agents constitute the bulk structure of the AgentService architecture.

The messaging subsystem provides the agents with a communication channel for message exchange. The default implementation is based on CLI remoting in order to provide this service. Different implementations can be installed: administrator of MASs might decide to rely on external messaging systems like Message Queuing systems in order to provide a wider set of service.

The persistency subsystem is responsible of saving and restoring the knowledge base of agents in case of system crash. The default implementation relies on object serialization and uses the file system as storage. High critical scenarios should require a more robust persistence system, i.e. a relational database.

The storage management system handles the installation of the assemblies containing the definition of agent templates, behaviour objects, and knowledge objects. Every time a new agent type is deployed on the platform, all the dependent assemblies need to be placed in the storage. When an agent is instantiated all the necessary information for its creation and its execution have to be found in the storage.

3.3. Agents Scheduling

Agents scheduling is one of the innovative features of AgentService: the scheduling model used by the platform guarantees the required autonomy to the agents and offers a real multi-threading context to the programmers. Scheduling of agents is based on CLI application domains: AgentService uses an application domain for each agent running on the platform while the multi-behavioural activity of agents is obtained by assigning one thread to each behaviour object used by the single agent. Concurrency and

race conditions among behaviours are avoided by using synchronization structures built inside AgentService.

Agent programming frameworks deal with the critical issue of scheduling by proposing two solutions: the creation of a separate process for each running agent and the use of one thread for each agent. The implementation of agents as operative system processes ensures isolation: each agent has a separate address space and resource sharing among processes is achieved by an explicit cooperation. Processes cannot be easily managed from other processes: only the operative system scheduler has complete control of their execution. An alternative solution is given by assigning one thread to each agent. The use of threads allows an easy management of agents: by using the common operative system APIs, agents can be started, stopped, resumed, and terminated. Threads do not guarantee the required isolation since threads in the same process share the same address space. Furthermore, threads cannot be given different execution permissions and they normally run with the same privileges of the owning process. Nonetheless, the common adopted solution for agent scheduling is the use of threads.

Application domains are a better solution for agents' scheduling since they solve many of the problems previously listed. Application domains are lightweight processes that live inside a process and they can run with user privileges different from the owning process and have a separate address space; the only way to communicate with them is using remoting. These features guarantee the required autonomy and isolation for each agent running. Application domains allow multi-threading: this characteristic gives the agents a real multi-behavioural activity. Finally, application domains can be managed as threads by the owning process and their setup is lighter than the process setup. The easy management of application domains allows the platform scheduler to control the agent activity in a smart and simple way. It is worth noting that the platform is the only entity that needs to control the activity of agents and the platform process is the only process that stores references to the dependent application domains. This does not break the isolation and the autonomy requirements of the agent programming model.

Many advantages gained with application domains can also be obtained by implementing ad-hoc infrastructures and layers of code that perform the missing functionalities. This task is time consuming and unnecessary when a development platform gives you the built-in components that perform the same work and they are completely integrated with the run-time environment. Application domains are fundamental elements within AgentService and simplify the scheduling system of the platform: most of the work is performed by the hosting operating system that reasonably offers a reliable service. AgentService also has a scheduling subsystem that controls the agents' activity and assigns to

them the required user privileges and the appropriate execution priority. Such a component relies on the CLI scheduler but custom implementations can be provided in order to give a more sophisticated scheduling policy. This is a critical task and a custom scheduler should be implemented only when it is very necessary.

3.4. Services and Interoperability

AMS, DF, and MTS provide services not only to agents running on the platform, but also to external clients. Interaction with other FIPA compliant platforms and legacy software is obtained by exposing these components by means of standard technologies and protocols as HTTP, HTML, and Web Services. AgentService relies on remoting in order to provide access to the FIPA components: AMS, MTS, and DF simply open a remoting channel to serve external requests. Ad-hoc adapters have been designed to provide a web interface and a web service front the end: these components simply act as a bridge between the different technologies, and make available platform services through worldwide standards. These elements are not components of the AgentService core and use the proprietary technology (i.e. ASP.NET) that is not part of the CLI. This architecture clearly separates the proprietary technologies from the AgentService portable core and it allows the use of third party implementations that can substitute the common components.

3.5. Language support

AgentService simplifies the definition of agent templates, behaviours, and knowledge types by using the Agent Programming eXtensions (APX) [13]. The Agent Programming eXtensions package provides the developer with a set of templates for the design and the implementation of software agents. APX also include a tailored compiler targeting agents to the AgentService platform. APX are built on top of the Common Language Infrastructure and they strongly rely on language interoperability. APX does not implement a general-purpose language: there are no first class types but run-time managed templates that can be programmed by the agent developer in a C#-like syntax. The advantages obtained by using APX are the following:

- APX have a clear agent oriented interface: the key elements of the agent model are exposed with specific constructs that are part of the language;
- APX have a semi-automatic handling of concurrency: knowledge objects can only be accessed inside synchronization blocks and are not visible outside of these blocks;
- APX offer exactly what the agent designer needs: the developer can only define agent templates, knowledge types, and program behaviours and rely on the other programming languages to define the object oriented aspects of the software project. Programmers cannot design illegal code (i.e.: storing references to agents);
- APX syntax is very similar to C#-syntax: the grammar for expression and statements is almost identical to the C# one. Few productions have been added and they are used to express the key elements of the underlying agent model;
- APX are a full client of the object-oriented model: each component designed for the CLI can be imported and instantiated as in any other CLI compliant programming language.

The reduced grammar of APX can appear incomplete: APX defines templates and does not allow the management of these templates inside the language. This is a particular design choice that clearly reflects the underlying agent model: the programmer does not have to manage the types it defines since this task is performed by the platform. Furthermore, APX does not allow the definition of classes or interfaces but leverage on language interoperability to have full access to object-oriented model defined in the CLI. This aspect underlines the clear agent oriented interface of the extensions: APX have been designed to work in synergy with the other programming languages while keeping the two programming models separate.

4. Conclusions

This paper presented AgentService an agent oriented framework. The key factors of AgentService are the agent model, the scheduling of agents and its modular architecture. While many agent-oriented frameworks adopts Java as implementation technology, AgentService is built on top of the Common Language Infrastructure and it exploits all innovative features of CLI. In particular, the use of application domains and remoting has been a successful design choice, which has favoured the implementation of a flexible and reliable run-time structure for the agents hosted in the multi-agent system.

AgentService has been designed in order to be portable on different operating systems and platforms. Thanks to its modular architecture, it can be tailored to exploit the specific advantages of the hosting operating system. Furthermore, the Agent Programming eXtensions offer an effective way to design and to implement agents. The full integration with CLI and the use of language interoperability, make APX offer a powerful programming environment by maintaining a clear agent oriented interface.

All these features make AgentService a valuable agent-programming framework that supports the software developer in every phase of the software project: from the design to the deployment.

References

- [1] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, Reading, Massachusetts, 1994.
- [2] C. Szyperski, "Component Software - Beyond Object-Oriented Programming" 2nd Ed., Addison-Wesley, ACM Press, 2002.
- [3] M. Wooldridge, N. R. Jennings, "Intelligent Agents: Theory and Practice", The Knowledge Engineering Review, 10(2):115-152, 1995.
- [4] S. Franklin, A. Graesser, "It is an Agent, or just a Program?: A Taxonomy of Autonomous Agents", Proceedings of the Third International Workshop of Agent Theories, Architectures, and Languages, Springer-Verlag, 1996.
- [5] G. Weiss, "Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence", G. Weiss Editor, MIT Press.
- [6] Foundation of Intelligent Physical Agents (FIPA), <http://www.fipa.org>
- [7] Standard ECMA-335: Common Language Infrastructure (CLI) 2nd Editino, Dec. 2002, ECMA, available at: <http://www.ecma-international.org/publications/standards/Ecma-335.htm>.
- [8] Standard ISO/IEC 23271:2003: Common Language Infrastructure, March 28, 2003, ISO.
- [9] Microsoft Shared Source CLI, available at: <http://msdn.microsoft.com/library/default.asp?url=/msdnmag/issues/02/07/sharedsourcecli/toc.asp>
- [10] A. Rao, M. Georgeff, "BDI Agents from theory to practice" in Proc. 1st Int. Conf. on Multi-Agent Systems (ICMAS-95), San Francisco, USA, April 1995.
- [11] A. Rao, M. Georgeff, D. Kinny, "A methodology and modelling technique for systems of BDI Agents" in Proc. 7th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAA-MAW'96), January 22-25, 1996, Eindhoven The Netherlands.
- [12] FIPA Abstract Architecture Specification, available at <http://www.fipa.org/specs/fipa00001/>
- [13] C. Vecchiola, M. Coccoli, A. Boccalatte, "Agent Programming Extensions relying on a component oriented infrastructure", Proceedings of the 2003 IEEE International Conference on Information Reuse and Integration (IRI - 2003), Oct. 26-29, 2003, Las Vegas, NV.

An Analytical Framework for Consistency Maintenance Mechanisms in Collaborative Editing Systems

Liyin Xue
Dept of Computing
Macquarie University
Sydney, NSW 2109 Australia
lyxue@ics.mq.edu.au

Mehmet Orgun
Dept of Computing
Macquarie University
Sydney, NSW 2109 Australia
mehmet@ics.mq.edu.au

Kang Zhang
Dept of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688, USA
kzhang@utdallas.edu

Abstract. This paper proposes an analytical framework for consistency maintenance mechanisms in collaborative editing systems. Unlike the existing frameworks, it takes into account the involvement of human users in the consistency maintenance process. The framework classifies consistency maintenance mechanisms into four categories, i.e., rule-based automatic conflict prevention, conflict prevention by user-centred meta-negotiation, rule-based automatic conflict resolution, and conflict control by negotiation. It suggests some new consistency maintenance issues in Internet-based real-time collaborative editing environments.

1. Introduction

With the development of the Internet, groupware systems have become more widespread. Collaborative editing systems or group editors are a particular type of groupware systems. They are designed to support multiple users to jointly edit, annotate, and revise a shared document. The goal is to support the cooperation and collaboration processes between co-authors [28].

In terms of time dimension, collaborative editing systems can be classified into synchronous (or real-time) and asynchronous ones [12]. Asynchronous collaboration occurs when co-authors work on the same document at different times. A real-time group editor is a system that allows multiple users to simultaneously edit a document without the need for physical proximity [25].

In terms of system architecture, there are three categories of collaborative editing systems: centralised, replicated, and hybrid [8]. Generally, centralised systems are easier to implement and particularly suitable for tightly-coupled collaboration in local area network (LAN) environments. The replicated or hybrid systems have a more responsive interface but need more effort to implement. They are suitable for loosely-coupled collaboration over the Internet.

Since a shared document is under the editing of multiple users, consistency maintenance becomes a major issue to the system designers. In a loosely-coupled, distributed, real-time group editing environment based on the Internet, the What You See Is What I See (WYSIWIS) property cannot be guaranteed, that is, each user cannot instantly see changes made by other users to shared objects, so users may unknowingly happen to edit the same part of the document at

the same time [22]. Therefore, the major challenge of consistency maintenance is the management of multiple streams of concurrent activities so that document consistency can be maintained in the event of conflicts [9]. Many existing systems use various techniques (e.g. locking) invented for database systems to *prevent* such a contention [13, 22]. However, the conflict prevention strategy of database systems is in contradiction to the spirit of collaboration in groupware systems. A consistency maintenance mechanism must facilitate the discussion and negotiation process amongst the involved users rather than inhibit the expression of their different opinions or intentions.

There have been some efforts in proposing innovative techniques for consistency maintenance in real-time collaborative editing systems, which take into account human users' intentions and are particularly suitable for wide area network environments, like the Internet [6, 27, 30, 31, 33, 34, 35, 36, 38]. However, it is not clear how these techniques are related to the traditional ones invented for database systems. It is necessary to have an analytical framework to systematically examine the existing consistency maintenance mechanisms, which may shed light on new directions for the exploration of new mechanisms.

The rest of the paper is organised as follows. Section 2 proposes our new taxonomy of consistency maintenance mechanisms. Based on it, four categories of consistency maintenance mechanisms are systematically examined from section 3 to 6. Finally, Section 7 compares our work with related work and summarises the major contributions of this paper.

2. An Analytical Framework

Consistency maintenance is a well-studied topic in distributed systems, mobile systems, and database systems. Generally, the consistency maintenance mechanisms implemented in these systems are system-oriented in the sense that consistency is predetermined or automatically enforced by the systems without the involvement of end users. Many research prototypes of groupware systems simply adopted the mechanisms implemented in these systems. In order to provide support for the situated negotiation among participating users, some prototypes took into account human user intentions [20, 32, 38]. Therefore, our first dichotomy of consistency maintenance mechanisms is users' *involvement in*

or *detachment* from the maintenance processes. Conceptually, it captures the fundamental difference between database systems and collaborative systems. Technically, it is useful in classifying the existing consistency mechanisms that are either *system-oriented* or *user-centred*.

Theoretically, we can reach a unified view of consistency by arguing that *consistency is an agreement (or inter-subjectivity) among human users* in groupware systems [37]. With such a user-centred interpretation, the *system-predetermined consistency* is simply an agreement reached in advance (i.e., before the ongoing collaborative work) among the group members. In contrast, the agreement reached after situated negotiation among the users is referred to as *emergent consistency*. From this perspective, the end users' actions and interaction become the focus of our examination.

A single-user editor provides the users with interaction functions such as tailoring (configuring), browsing, editing, and annotating, etc. Each of them has a corresponding category of user actions. More importantly, multi-user editors must deal with communication and collaboration aspects of interaction in addition to those supported in single-user editors. We introduce two actions that are more group oriented, i.e., *meta-negotiation* and *semantic-negotiation*, to model these aspects of user actions.

All user actions could be broadcast to remote sites. However, a user may not be allowed to activate a particular function of an editor. By meta-negotiation we mean that users may not have been granted the necessary right to browse, edit, or annotate particular part of the data, or to tailor specific functionality of the application, thus they need to negotiate with others in order to have the right before they can activate the functions. For example, access control is an example of meta-negotiation. Whether a user has the right to access a data item is pre-determined by the system administrators or its owner in many groupware systems. The meta-negotiation functions are defined and implemented in advance, and configured by system administrators. Meta-negotiation could be user-centred in the sense that the access right is a result of negotiation between users during the ongoing cooperation process. For example, if a data item is locked, a user may activate a meta-negotiation command to ask whether the current lock owner can release his/her lock. It is up to the lock owner to make the decision.

Semantic-negotiation is an act of message exchanges by various user actions or their combinations in order to reach certain agreement among users, whereas meta-negotiation is about the right to activate a particular action. Obviously, semantic-negotiation is a higher-level user action, which may involve various other user actions. Negotiation may happen as users browse, modify, or annotate the data.

Now it is clear that consistency maintenance mechanisms are functions that a system provides to support meta-negotiation and semantic-negotiation among end users. These functions may be automatically enforced by the system or explicitly invoked by the end users.

Concurrency control is generally used to *prevent* inconsistent concurrent updates in database and distributed systems. This is the major meaning of the concept in the literature, though it is sometimes intended to mean the general consistency maintenance. We will use it in the more restrictive

sense. Meta-negotiation is related to the coordination of users' access to the data or the functionality of the system in the course of cooperative working. It can be supported by access control and concurrency control mechanisms.

Since conflict prevention approaches are inherently restrictive, some cooperative editing systems dispense with any concurrency control altogether [12, 29]. In fact, these systems rely upon social protocols and users' awareness of others' actions to prevent conflicts, and hope that if conflicts do occur, they can be quickly and easily resolved. However, since the WYSIWIS property cannot be maintained in an environment with a non-deterministic communication latency, such as the Internet, preventing conflicts based on awareness may not function well.

Without proper conflict prevention, conflicts seem inevitable, and supporting mechanisms are needed to help users resolve them. Conflict regulation, conflict resolution, conflict control, and conflict management have been used to mean consistency maintenance in groupware literature, particularly in the area of asynchronous groupware systems. We restrict them to mean mechanisms supporting human users in resolving conflicts due to concurrent activations of some functions of a collaborative editing system by different users, or due to alternative intentions of multiple users to act upon the same part of data. Semantic-negotiation can be supported by conflict control mechanisms.

Based on the above discussion, our second dichotomy for the classification of consistency maintenance mechanisms can be identified as *conflict prevention* and *conflict control*. Consistency maintenance covers not only conflict prevention but also conflict control.

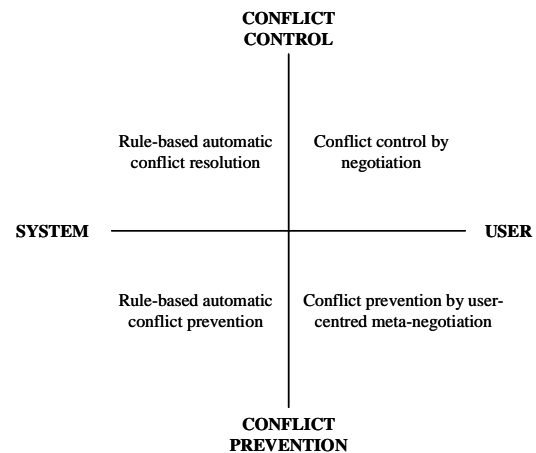


Figure 1 Taxonomy of consistency maintenance mechanisms

With the above two dimensions, i.e., system-user and prevention-control, we can classify consistency maintenance mechanisms into four categories: rule-based automatic conflict prevention, conflict prevention by user-centred meta-negotiation, rule-based automatic conflict resolution, and conflict control by negotiation. The taxonomy is illustrated in Figure 1. Generally, on the one hand, with the absence of human user's involvement, the system prevents or resolves conflicts automatically; on the other hand, the system may

provide some support to facilitate human users in preventing or controlling conflicts in the coordination and cooperation processes. In the following four sections, we will examine the four categories of consistency maintenance mechanisms in detail by illustrating some representative ones. It is worth noting that our examination focuses on the analytical framework rather than the individual mechanisms.

3. Rule-based Automatic Conflict Prevention

Access control and locking are the most widely used rule-based automatic conflict prevention approaches in both database systems and groupware systems [2, 13, 19]. They are well-understood mechanisms in terms of both implementation and system usage. There are some other variants based on them. Conflicts are prevented automatically by the system according to the rules configured in advance.

Predetermined turn-taking: With the predetermined turn-taking approach, only one user at a time has the “token/floor” to edit the shared data or to hold the group pointer (e.g., in the case of computer-based teleconferencing). Access to the token/floor is controlled by internal technical protocols. The system passes the token/floor to each user according to pre-specified rules (e.g., time-sharing scheme). Users are in a passive state controlled by the system. Obviously it does not support concurrent user actions.

Locking: With the locking approach, a shared data item is first locked under the implicit or explicit request of a user before it is updated [13]. So only one user at a time is able to update it. Conflicts can be prevented by locking, but locking is undesirable because it interrupts users in their work, inhibits collaboration, and causes unnecessary overhead in collaborative systems. The overhead of requesting and releasing locks will increase response time of user actions in distributed environments.

4. Conflict Prevention by User-centred Meta-negotiation

With the conflict prevention by user-centred meta-negotiation approaches, there is no more than one user editing the shared data item at any time, and the right to have access to the data item is a result of negotiation among the involved participants. The central point of user-centred meta-negotiation is how the process of negotiation for access to data is supported.

Negotiable turn-taking: Unlike the predetermined turn-taking, access to the token is controlled by external social protocols in the negotiable turn-taking approach. For example, whenever a user wants to have access to a data item, s/he needs to negotiate with the coordinator or current token holder. Similar to predetermined turn-taking, it is not suitable for collaboration sessions with much parallelism among participants. It may overly inhibit the free and natural flow of information among participants [10]. However, it is suitable for and widely used in audio and video conferencing systems.

Negotiability: It is a class of mechanisms that support the negotiation between a requestor (or activator) who wants to use an application function and a coordinator (or affected user) by offering a technical channel of communication and by

allowing the affected user to intervene against the decision of the activator by technical means [34]. The negotiability approach is mainly for the negotiation between two users involved in an asynchronous groupware system. To explicitly negotiate on the right to activate a function is time consuming if it involves multiple users in synchronous environments.

5. Rule-based Automatic Conflict Resolution

With rule-based automatic conflict resolution approaches, users are allowed to edit the shared data freely, and any conflict will be automatically resolved by the system in terms of pre-determined rules. They are different from each other in how a conflict is automatically resolved.

Centralised ordering (pessimistic serialisation): Assume that data items are replicated over all user workstations, and there is a central controller process (server) sitting on one of them. The server receives user requests for operations and distributes the operations received to every client (including the one that issued the operations). Since the requests are always granted, this approach simply imposes a global ordering on the operations issued by different users, i.e., the operations are performed in the same order at all participating sites. It has been implemented in many systems such as *Villa* [3]. The major drawback of this approach is its irresponsiveness if the communication latency is high, since an operation is executed only after it is received from the server rather than when it is requested, i.e., pessimistic execution.

Decentralised ordering (optimistic serialisation): Unlike the pessimistic scheme, operations issued at a site are immediately executed at the local site in the decentralised ordering approach. The operations are usually time-stamped using logical clocks such that a global order can be enforced [17, 26]. The requests for ordering as required in the centralised ordering scheme are not necessary. When two or more semantically conflicting operations have been executed concurrently, one (or more) of these operations is undone and re-executed in the correct order such that replica convergence is guaranteed. This serialisation mechanism is implemented in GroupDesign [15] and LICRA [14]. Karsenty and Beaudouin-Lafon use information about commutativity and masking of operations to minimise the number of undo operations in their ORESTE algorithm [16]. This approach is very responsive (in the local user interface).

Operational transformation: Similar to the optimistic serialisation approach, when an operation is requested, the local editor performs the operation immediately, and then multicasts it to the remote sites. However, with the operational transformation approach, when an operation is received from a remote site, the local editor needs to check whether there are concurrent operations executed, if so, the operation needs to be *transformed* such that the requestor’s original intention is preserved [10, 30]. The primary advantage of this approach is that users’ concurrent intentions are preserved if they are not conflicting with each other [36].

The major problem of the rule-based automatic conflict resolution mechanisms is intention violation. Only one of the concurrent conflicting intentions will be preserved. For example, if two users concurrently change the colour of an object to red and green respectively, the final execution effect

will be either red or green depending on the global order defined in the serialisation approaches, thus the intention of one of the two users will be violated.

6. Conflict Control by Negotiation

Automatic conflict resolution has the advantage of being efficient. Nevertheless, it is generally infeasible for the system to have the knowledge to properly resolve conflicts among users. Conflicts are best resolved by end users, with the system providing explicit information about users' actions and negotiation support mechanisms. The question now is what supporting mechanisms the users need. They depend on the characteristics of the conflict, the effort the users would like to take, the availability of communication channels, and the prestigious status of individual users in the decision process. There are too many parameters that can affect the process of conflict resolution. Consequently, negotiation support mechanisms can be classified in various ways. Here, we present different mechanisms of negotiation support in terms of their roles in different stages of a negotiation process, that is, from conflict notification, mediation support, coordination support, to final decision support. The mechanisms for each stage are further classified. A negotiation support system may employ a combination of these mechanisms.

6.1. Conflict Notification and Visualisation

Supporting responsiveness, any lazy consistency approach (without conflict prevention) contains inherent race conditions. So collaborative editing systems need to detect conflicts after they occur and make the users be aware of them. These are the basic functionality of conflict awareness mechanisms that differ from each other in what to report to and how to notify the involved users.

Temporal conflict and simple notification: Stefik et al use time-stamps of operations to detect conflicts between them, which are then resolved manually by the involved users [29]. The conflict is defined at time-stamp level without operation semantics being taken into account. How a conflict is notified is not clear from their paper. For a simple notification, a popup message can be delivered on the screens of the involved users, which describes the place of the conflict in the document. The drawback is the users have to locate the conflict and identify its nature.

Syntactic or semantic conflict and conflict visualisation: If the system can use operation semantics to identify the nature of conflicts and present them visually, the involved users can then focus on how to resolve them. *Diffing* approaches have been widely used in document merging [21]. With interactive diffing, the system can point out the differences between two versions of a document and ask for the user to make decisions. The PREP system's *flexible diff* can visually report conflicts in various granularities such that users can detect the conflicts without being distracted from more appropriate tasks [24].

Any system supporting negotiation must provide some conflict notification or visualisation mechanism. For example, all real-time graphics editors supporting multi-versioning

visualise conflicts in multiple versions [6, 20, 36, 38]. Conflict detection and visualisation are important issues that need to be further studied in real-time collaborative editing systems.

6.2. Mediation Support

Being aware of the conflict, the involved users will start a negotiation process to resolve it. Negotiation can be mediated in different ways in terms of the communication media and the organisation of messages. However, we will focus only on the visible referential artifacts of the negotiation process, which are shared by the involved users. A referential artifact could be the base document or any annotation to it. With the former, the users may be allowed to edit the document during the process of negotiation, or to browse it first and then update it only when they have reached a consensus via an extra communication channel (e.g., voice). With the latter, the content of the referential artifact may be a new version or simply some comments on the base document. We categorise three alternatives of mediation support mechanisms as follows:

Commenting annotation: Annotation is widely used in asynchronous group work. For example, PREP [23] and Quilt [18] are well-known asynchronous systems supporting annotation. Recently, there are a number of researches on web annotation for supporting group work [5]. In many real-time groupware systems, commenting annotation mechanism is usually implemented as a voice channel. A popup message window is also useful. For example, the *Anchored Conversations* prototype provides a synchronous text chat window that can be anchored to a specific point within a document, and moved around like a post-it note [7].

Intrusive negotiation: It is a common practice that people discuss with each other about a document on a piece of paper or about a figure on a whiteboard in the same place and at the same time. They can modify the document or the figure while the discussion is in progress. This is also possible in real-time distributed group editors if people would follow the required social protocols of negotiation. The drawback is that it may make the document become "dirty" due to the nature of unpredictable negotiation dynamics. There are some group editors supporting intrusive negotiation by embedding multiple versions of the same object in conflict into the document, for example, Tivoli [20] and GRACE [6, 32]. The versions created are mixed up with the base document. The advantage is that all individuals' intentions are visualised, thus facilitating the negotiation process. However, the embedded versions change the context of the object in conflict. This may confuse the users. The following alternating annotation mechanism can solve this problem.

Alternating annotation: If multiple authors disagree with each other on a particular chunk of the document, each of them can propose her own version but none of them will be included into the final document before a consensus is reached. Each user can edit her own version while the negotiation is in progress. Therefore the document is annotated with versions rather than embedded with them. The Quilt system supports both commenting annotation and alternating (called revision) annotation in an asynchronous way [18]. To the best of our knowledge, POLO is the only real-time group editor implementing such a mechanism [38].

Obviously, mediation support is related to the issue of multi-user interface design that is a challenging issue in real-time group editors. The way in which a mediation support mechanism can be integrated into the general editing process is an interesting direction for further research.

6.3. Coordination Support

Concurrency control is employed to prevent potential conflicts from happening. With unconstrained and responsive editing without conflict prevention, if conflicts occur, coordination mechanisms may be needed to harness them such that they do not become unmanageable. Therefore, concurrency control (or coordination support for conflict prevention) and coordination support for negotiation are rather similar. They differ in that the former prevents any conflict to occur before users make any change to the data, whereas the latter prevents any unmanageable conflict to occur or controls further conflicts after users have been involved in conflicts.

Unconstrained multi-versioning: Multi-versioning mechanisms provide a basic level of coordination that guarantees a convergent document state for all users and preserves all individuals' concurrent conflicting intentions [6, 20]. They are extensions of conventional version control mechanisms in real-time and multi-replica environments. An unconstrained multi-versioning scheme is able to support unconstrained editing. The price it pays is a higher degree of difficulty of semantic conflict resolution. The users rely on social protocols to coordinate their conflict resolution process.

Predetermined post-locking: Post-locking is a class of mechanisms for coordinating users in resolving the conflicts which result from unconstrained accesses to shared data [35]. Only when a conflict occurs will the system automatically lock the object in conflict. It is different from the well-studied conflict prevention lock, which we call pre-lock. The system can assign the lock ownership of the created versions to corresponding users. However, the assignment schemes are static, i.e., predetermined by the system configuration.

Negotiable lock transferring and assigning: Generally, mechanisms which involve the users are appropriate and valuable in groupware applications. Similar to the negotiability mechanisms, post-locking can be made dynamic or negotiable. For example, if there are two users involved in a conflict, the system may first give the right of modifying the object to one of the users. Later on, the two users may negotiate with each other on the access right to the object. Here, the negotiable turn-taking mechanism may be applicable. If the multi-version approach is employed, users may transfer lock ownerships on different versions dynamically such that group discussion and negotiation are smoothly facilitated. This topic is worthy of further investigation. It may open up a new research direction for applying the conventional concurrency control mechanisms in a new context.

6.4. Decision Support

Decision support is a grand concept that may also cover all the steps of negotiation we have just discussed. However, we restrict it to mean activities involved in the final step of

negotiation, when several alternatives for the resolution of the conflict may have been proposed for a final decision after rounds of discussion and negotiation. The final decision may be made by a representative or by all the involved users via voting.

Representative decision: This is a very efficient way of conflict resolution. Only one user represents the group to resolve the conflict, or to execute the agreed upon group intention. The GINA system supports synchronous group editing [1]. When conflicts occur, it provides a selective undo/redone mechanism for one of the involved users to resolve them.

Participatory decision: With participatory decision or voting, each involved user has a say on the final result. Although it is more democratic than the representative decision, it may be less efficient. The POLO system implements such a mechanism [38].

7. Comparison and Conclusion

Many reviews on consistency maintenance mechanisms have been presented in the groupware literature. Unfortunately, some of them simply gave a list of some of the mechanisms without pointing out their relationships, though they captured their respective characteristics [11, 16, 22, 31]. Others did provide analytical frameworks to examine the consistency mechanisms. However, they are only applicable to a subset of the mechanisms having been examined in this paper. In other words, they cover only one or two of our four categories. For example, Greenberg et al provided a taxonomy for various locking and serialisation approaches (i.e., some of the mechanisms in the categories of rule-based automatic conflict prevention and rule-based automatic conflict resolution) [13]. Wulf classified conflict regulation mechanisms for asynchronous groupware systems, which are mainly in our conflict prevention by user-centred meta-negotiation category [34]. Bhola et al presented a taxonomy for a set of ordering mechanisms, which are those mechanisms in our rule-based automatic conflict resolution category [3, 4]. Sun and Ellis systematically analysed existing operational transformation schemes [30]. Obviously, none of them covered the consistency mechanisms comprehensively, although they are complementary to our work.

In summary, we have proposed an analytical framework for the examination of consistency maintenance mechanisms in collaborative editing systems, which is comprehensive and able to capture the fundamental characteristic of groupware systems, that is, the involvement of human users. Although we analyse all the four categories of consistency maintenance mechanisms, we put more efforts on the conflict control mechanisms, which are different from conventional conflict prevention ones and have the potential to support Internet-based unconstrained real-time collaborative editing. In particular, we have examined the conflict control by negotiation category in detail. The examination suggests some new consistency maintenance issues, such as, how to detect and visualise conflicts, and how to mediate and coordinate a negotiation process, in real-time collaborative editing environments. None of the existing analytical frameworks covered this category of mechanisms. Finally, although our

focus is on collaborative editing systems, the framework presented in this paper is general and applicable to other groupware systems.

References

- [1] T. Berlage. A selective undo mechanism for graphical user interfaces based on command objects. In *ACM Transactions on Computer-Human Interaction*, 1(3), 1994, pp. 269-294.
- [2] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency control and recovery in database systems*. Addison-wesley, Reading MA, 1987.
- [3] S. Bhola and M. Ahamad. The design space for data replication algorithms in interactive groupware. In *Georgia Institute of Technology Technical Report GIT-CC-98-15*, 1998.
- [4] S. Bhola, G. Banavar, and M. Ahamad. Responsiveness and consistency tradeoffs in interactive groupware. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, Nov. 1998.
- [5] J.J. Cadiz, A. Gupta, J. Grudin. Using web annotation for asynchronous collaboration around documents. In *Proceedings of ACM Conference on CSCW*, Dec. 2000, pp.309-318.
- [6] D. Chen and C. Sun. A distributed algorithm for graphic objects replication in real-time group editors. In *Proceedings of the ACM Conference on Supporting Group Work*, Nov. 1999, pp.121-130.
- [7] E. F. Churchill, J. Trevor, S. Bly, L. Nelson, and D. Cubranic. Anchored conversations: chatting in the context of a document. In *Proceedings of the ACM CHI'2000*, The Hague, Amsterdam, pp.454-461.
- [8] P. Dewan. Architectures for collaborative applications. In M. Beaudouin-Lafon (ed.), *Computer Supported Cooperative Work*, John Wiley & Sons, 1999, pp.169-193.
- [9] P. Dourish. Consistency guarantees: exploiting application semantics for consistency management in a collaboration toolkit. In *Proc. of the ACM Conference on CSCW*, Nov. 1996, pp.268-277.
- [10] C.A. Ellis and S.J. Gibbs. Concurrency control in groupware systems. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, May 1989, pp.399-407.
- [11] C.A. Ellis, S.J. Gibbs, and G.L. Rein. Groupware: some issues and experiences. In *Communications of ACM*, 34(1), Jan.1991, pp.39-58.
- [12] S. Greenberg, M. Roseman, D. Webster, and R. Bohnet. Issues and experiences designing and implementing two group drawing tools. In *Proceedings of the 25th Annual Hawaii International Conference on the System Sciences*, January 1992, pp.139-250.
- [13] S. Greenberg and D. Marwood. Real time groupware as a distributed system: concurrency control and its effect on the interface. In *Proceedings of the ACM Conference on CSCW*, Oct. 1994, pp.207-217.
- [14] Kanawati. LICRA: A replicated-data management algorithm for distributed synchronous groupware application. In *Parallel computing*, vol. 22, 1997, pp.1733-1746.
- [15] A.Karsenty, C. Tronche, and M. Beaudouin-Lafon. GroupDesign: shared editing in a heterogeneous environment. In *Usenix Journal of Computing Systems*, 6(2), 1993, pp.167-195.
- [16] A.Karsenty and M. Beaudouin-Lafon. An algorithm for distributed groupware applications. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, May 1993, pp.195-202.
- [17] L. Lamport. Time, clock, and the ordering of events in a distributed system. In *CACM* 21(7), July 1978, pp.558-565.
- [18] M.D.P. Leland, R.S. Fish, R.E. Kraut. Collaborative document production using Quilt. In *Proceedings of the ACM Conference on CSCW*, 1988, pp. 206-215.
- [19] A.Michailidis and R. Rada. A review of collaborative authoring tools. In R. Rada (ed), *Groupware and Authoring*, Academic Press Limited, 1996, pp.9-44.
- [20] T. P. Moran, K. McCall, B. van Melle, E. R. Pedersen, and F.G.H. Halasz. Some design principles for sharing in Tivoli, a white-board meeting support tool. In S. Greenberg, S. Hayne, and R. Rada (eds.), *Groupware for Real-time Drawing: A Designer's guide*, McGraw-Hill, 1995, pp.24-36.
- [21] J. P. Munson and P. Dewan. A flexible object merging framework. In *Proceedings of the ACM Conference on CSCW*, Oct. 1994, pp.231-242.
- [22] J. P. Munson and P. Dewan. A concurrency control framework for collaborative systems. In *Proceedings of ACM Conference on CSCW*, 1996, pp. 278-287.
- [23] C.M. Neuwirth, D. S. Kaufer, R. Chandhok, and J. H. Morris. Issues in the design of computer support for co-authoring and commenting. In *Proceedings of the ACM Conference on CSCW*, Nov. 1990, pp.183-193.
- [24] C.M. Neuwirth, R. Chandhok, D. S. Kaufer, P. Erion, J. Morris, and D. Miller. Flexible diff-ing in a collaborative writing system. In *Proceedings of the ACM Conference on CSCW*, Nov. 1992, pp.147-154. Also in R. Rada (ed.): *Groupware and Authoring*, Academic Press Limited, 1996, pp.189-204.
- [25] A.Prakash. Group editors. In M. Beaudouin-Lafon (ed.), *Computer Supported Cooperative Work*, John Wiley & Sons, 1999, pp.103-133.
- [26] M. Raynal and M. Singhal. Logical time: capturing causality in distributed systems. In *IEEE Computer Magazine*, 29(2), Feb. 1996, pp.49-56.
- [27] M. Ressel and R. Gunzenbaeuser. Reducing the problems of group undo. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work*, Nov. 1999, pp.131-139.
- [28] T. Rodden. A survey of CSCW systems. *Interacting with Computers*, 3 (3), 1991, pp.319-353.
- [29] M. Stefik, G. Foster, D. G. Bobrow, K. Kahn, S. Lanning, and L. Suchman. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. In *Communications of ACM*, 30(1), 1987, pp. 32-47.
- [30] C. Sun and C.A. Ellis. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In *Proceedings of ACM Conference on Computer Supported Cooperative Work*, Nov. 1998, pp.59-68.
- [31] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality-preservation, and intention-preservation in real-time cooperative editing systems. In *ACM Transactions on Computer-Human Interaction*, 5(1), March 1998, pp.63-108.
- [32] C. Sun and D. Chen. A multi-version approach to conflict resolution in distribute groupware systems. In *Proceedings of International Conference on Distributed Computing Systems*, April 2000.
- [33] N. Vidot, M. Cart, J. Ferrie, and M. Suleiman. Copies convergence in a distributed real-time collaborative environment. In *Proceedings of ACM Conference on CSCW*, Dec. 2000, pp.171-180.
- [34] V. Wulf. On conflicts and negotiation in multiuser applications. In A. Kent and J. G. Williams (eds.): *Encyclopedia of Microcomputers*, Marcel Dekker, New Basel, Vol 23, Suppl. 2, 1999, pp. 63-88.
- [35] L. Xue, K. Zhang, and C. Sun. Conflict control locking in distributed cooperative graphics editors. In *Proceedings of the 1st International Conference on Web Information Systems Engineering (WISE' 2000)*, Hong Kong, IEEE CS Press, June 2000, pp.401-408.
- [36] L. Xue, M. Orgun, and K. Zhang. Intention preservation by multi-versioning in distributed real-time group editors. In *Proceedings of The International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS' 2002)*, Beijing, China). *Lecture Notes in Computer Science*, Vol. 2480, 2002, pp.510-524.
- [37] L. Xue, M. Orgun, and K. Zhang. A user-centred consistency model in real-time collaborative editing systems. In *Distributed Communities on the Web (DCW' 2002)*, *Lecture Notes in Computer Science*, Vol.2468, 2002, pp.138-150.
- [38] L. Xue, M. Orgun, and K. Zhang. Group-based time-stamping scheme for the preservation of group intention. In *DCW' 2002, Lecture Notes in Computer Science*, Vol.2468, 2002, pp.125-137.

An Experience of Fuzzy Linear Regression applied to Effort Estimation

Gerardo Canfora, Luigi Cerulo, Luigi Troiano
RCOST — Research Centre on Software Technology
Department of Engineering - University of Sannio
Viale Traiano - 82100 Benevento, Italy
{canfora, lcerulo, troiano}@unisannio.it
<http://cise.rcost.unisannio.it>

Abstract

Predicting the exact quality index number in software assessment is often not necessary. Software managers, especially in early phases, prefer to deal with fuzzy data such as intervals, ranges, and orders of magnitude. Possibilistic models may be used as a viable alternative to allow this type of analysis. This paper reports an experience of the application of fuzzy linear regression for predicting the maintenance effort of a software project.

1 Introduction

Effort estimation is a key aspect of risk management in software industry, and its assessment is usually addressed by predictive models. Statistical regression analysis, both linear and non linear, is widely adopted for predicting the effort on past projects data. However, Statistical Linear Regression (SLR) analysis works fine if data meets certain assumptions [17], such as:

- The distribution of residuals is normal;
- The variance of the residuals is constant with regard to every set of values for the independent variable;
- The error term is additive
- The expected (mean) value of the residuals is zero
- The mean of the error term is zero
- The expected correlation between residuals is zero
- Independent variables are uncorrelated with the error term
- No independent variables are a perfect linear function of other independent variables

Unlike other engineering disciplines, in Software Engineering (SE), as it is inherently knowledge intensive, these assumptions can be violated. In particular, normality of error distribution and constant variance of residuals could be not verified. Limitations of SLR have been discussed in literature and alternative solutions stemming from the area of Computational Intelligence have started to be investigated. For example [7] uncertainty and causal modelling for risk assessment are addressed using bayesian networks, and in [18] a fuzzy non linear regression technique is used to predict software faults. Other regression model approaches have been proposed in literature [5]. Among these, there is the Fuzzy Linear Regression (FLR) method.

Fuzzy regression was introduced by Tanaka et al. [16] to model situations in which the practitioner cannot accurately measure the dependent variable. It is a nonstatistical method and the deviation between observed and estimated values are assumed to be dependent on the vagueness of parameters which govern the system structure, not on its measurement errors [10]. This assumption is coherent in Software Engineering where usually measures are error-free, and functional models are only possible approximation of data sets. In SLR analysis the optimal criterion for curve fitting is the minimization of the error, while in fuzzy regression is the minimization of the vagueness of the dependent variable. It has been stated that fuzzy regression may be more effective than statistical regression when the basic assumptions are violated, as for example, when human judgment are involved [16], ambiguous processes must be explained [8], and when only a small amount of data are available and the aptness of the regression model are difficult to justify [1]. This kind of considerations led us to investigate how FLR behaves with regard to effort estimation, comparing results with SLR analysis in a case study that have shown to be well analyzed by statistical approach.

This paper reports an initial experience of applying FLR analysis to an industrial case study aimed at building a prediction model for software production effort. The remainder

of the paper is organized as follows: Section 2 provides an overview of the FLR technique adopted in our experience; Section 3 describes the case study and reports results from the application of SLR; Section 4 discusses results from the application of FLR; Section 5 concludes the paper and presents some future directions.

2 An overview of Fuzzy Linear Regression

A linear model has the form

$$y = a_0 + a_1x_1 + \dots + a_nx_n \quad (1)$$

An SLR (crisp) model is modified as

$$y = a_0 + a_1x_1 + \dots + a_nx_n + \epsilon \quad (2)$$

where ϵ (known as *error term*) is a stochastic variable describing the discrepancy between data samples and predicted values. A FLR model assumes a fuzzy linear function as

$$Y = A_0 + A_1X_1 + \dots + A_nX_n \quad (3)$$

where independent variables X_1, X_2, \dots, X_n , or parameters A_0, A_1, \dots, A_n , or both, can be fuzzy numbers referring the uncertainty of data and vagueness of model.

In our experience, data are referred to factors precisely quantified such as the number of software code components (SC) or the number of candidate impacts (CI). Therefore, we will refer to crisp input data, and fuzziness is only considered with regards to parameters such as

$$Y = A_0 + A_1x_1 + \dots + A_nx_n \quad (4)$$

The problem that arises is how to determine fuzzy parameters. There several methods to approach the problem: fuzziness of model fitness minimization [16, 14, 3, 15]; least squares of errors [2, 6, 13, 4], interval analysis [9]. In our experience we considered the first approach.

The method adopted is aimed at finding a regression model that fits all data within a specified fitting criterion. According to Zadeh's extension principle [19], dependant variable Y is still fuzzy. Since, fuzzy numbers are described by *possibility distributions* [11], variable Y describes a set of linear models in possibility. Parameters take usually the form of symmetric triangular fuzzy numbers $A_i = (c_i, s_i)$ represented by the following membership function:

$$\mu_{A_i}(x) = \max\left(0, 1 - \frac{|x - c_i|}{s_i}\right) \quad (5)$$

in which c_i is the *central value* and s_i is the *spread* value. Consequently

$$\mu_Y(y) = \max\left(0, 1 - \frac{\left|y - \left(c_0 + \sum_{i=1}^n c_i x_i\right)\right|}{s_0 + \sum_{i=1}^n s_i |x_i|}\right) \quad (6)$$

The Decomposition Theorem [11], states that a fuzzy number can be fully described by the set of intervals $[a_\lambda, b_\lambda]$ each with an associate possibility level of λ (α -cuts).

Then, fuzzy coefficients A_1, \dots, A_n are determined so that Y has the minimum spread while satisfying a degree of belief h . The term $h \in [0, 1]$ can be regarded as the desired level of compatibility between the data and the model. This means that crisp data must be in the fuzzy spread of Y with the at least a value of possibility h . This means that Y assumes the possibility value h at the farthest data point. The higher is the level of confidence required, the wider is the fuzzy spread of Y .

Determination of parameters can be seen as an optimization problem. Let x_{1j}, \dots, x_{nj} are the j -th input values of independent variables. For a given degree of belief h the fuzzy regression algorithm determines the spreads and the center values of the parameters by resolving the following linear programming problem

$$J = \min \left\{ ms_0 + \sum_{j=1}^m \sum_{i=1}^n s_i |x_{ij}| \right\} \quad (7)$$

$$y_j \geq c_0 + \sum_{i=1}^n c_i x_{ij} - (1 - h) \left(s_0 + \sum_{i=1}^n s_i |x_{ij}| \right) \quad (8)$$

$$y_j \leq c_0 + \sum_{i=1}^n c_i x_{ij} + (1 - h) \left(s_0 + \sum_{i=1}^n s_i |x_{ij}| \right) \quad (9)$$

$$c_i \geq 0, i = 1, 2, \dots, n.$$

Given m data, there is an optimal solution for a given value h .

$$A_i^h = (c_i, s_i), i = 0, 1, \dots, n \quad (10)$$

A useful theorem states that an optimal solution for $h' \neq h$ can be obtained from the optimal h -level solution as

$$A_{h'} = \left(c_h, s_h \frac{1 - h}{1 - h'} \right) \quad (11)$$

3 Case study: adaptive maintenance of an industrial project

The case study we considered has been analyzed in [12] for predicting the maintenance effort by using the ordinary statistical linear regression technique. It stems from the past project data of a major international software enterprise, namely EDS Italia Software. From 1996 to 1999 EDS has conducted several Y2K and EURO projects by adopting an adaptive maintenance process based on a preliminary assessment phase aimed at decomposing an application portfolio into loosely coupled work-packets that could be independently and incrementally modified and delivered. The

application portfolio was composed of about 40,000 software components, including 7,082 COBOL programs and 6,850 JCL procedures, and was decomposed in 123 work-packets. The project started on January 2 1999 and finished on January 14 2000. The total effort spent was 457 man months. The average staff was 146 people. The maximum peak, reached in March 1999, was 179 people; altogether, 253 different people were employed. The project was conducted by maintenance teams distributed on three different sites.

Several metrics were collected during the different phases of the project. In particular:

- SC, as the number of software code components
- I, as is the number of candidate impacts
- AI, as the number of actual impacts; SAI as the number of actual standard impacts
- Effort, as the actual effort measured as man-days
- Staff, as the number of employed maintainers
- Duration, as the actual duration measured as number of calendar days

Table 1 reports some descriptive statistics of the collected metrics and Table 2 shows the parameters of three regression models as reported in [12]. The models were developed considering the metrics correlation matrix and regression analysis verifications.

Table 1. Descriptive statistics of the data set

Metric	Min	Max	Mean	StdDev
SC	2	1243	364.43	334.260
CI	8	6646	1109.84	1490.490
AI	1	504	40.306	91.278
SAI	0	461	25.209	76.340
NSAI	0	122	15.096	22.118
TC	1	9382	702.14	1876.950
Effort	3	278	56.654	54.244
Staff	2	27	6.95	4.375
Duration	3	127	27.925	20.465

Table 2 shows the performance of the models in terms of the determination coefficient R^2 , and adjusted determination coefficient $adjR^2$, which is an extension of R^2 taking into account the number of independent variables of the regression model. It assesses the capability of the model for fitting the sampled data and it is computed as:

$$R^2 = \frac{\sum_{i=1}^m (y_i^* - \bar{y})^2}{\sum_{i=1}^m (y_i - \bar{y})^2} \quad (12)$$

Table 2. Ordinary regression model parameters

model	coefficients	p-value	R^2	$adjR^2$
SC	0.141094	1.64E11	0.843	0.839
CI	8.925E-03	2.94E02		
sqrtSC	2.452531	2.52E10	0.915	0.914
sqrtAI	7.022846	5.76E10		
sqrtSC	2.262565	5.31E08	0.921	0.918
sqrtNSAI	4.660045	6.10E-04		
sqrtSAI	7.115134	3.65E05		

where y_i^* is the predicted value, y_i is the observed value, and \bar{y} is the mean of the observed values.

4 Experimental results

We have repeated the empirical study reported in [12] by applying the fuzzy regression analysis. The fuzzy models were obtained on the same data set and considering the same independent variables. Table 3 shows the fuzzy coefficients of the fuzzy linear equation 4 obtained with a compatibility level $h = 0.5$. A comparable assessment of the the fuzzy regression approach is difficult because performance indexes of statistical linear regression are correlated to their basic assumptions which are quite different from the fuzzy approach. For example, from equation 12, if y_i^* is a fuzzy number then R^2 is a fuzzy number and becomes difficult to compare it to the corresponding crisp number obtained from the statistical model. In order to make a comparable quantitative assessment of the performance of the fuzzy regression model we have *defuzzified* the output values by choosing the output central value; i.e. the value at possibility level equal to 1. Then in equation 12 y_i^* becomes c_i^* which is the central value of the output fuzzy number. Table 4 shows the performance in term of R^2 and $adjR^2$ computed in this way. Substantially, from R^2 performance perspective and for this particular case study, SLR and FLR have almost the same behavior. This is due to the fact that the quality of the collected metrics meets all the statistical regression assumptions.

Although the performance makes both methods seem similar, there are some qualitative differences that make FLR peculiar. Figures 2, 1, and 3 shows for each model the behavior of the fuzzy output variable in correspondence to each observed data point. On the y -axis is reported the effort level, while on the x -axis is reported a set of 26 data point in ascending order with the observed effort. At each data point the output fuzzy number is represented with the central value and the corresponding spread extremes for two different compatibility levels ($h = 0.7$ and $h = 0$). The

Table 3. Fuzzy regression model parameters (h=0.5)

model	coefficients
SC	(68.72593, 323.16480)
CI	(0.38862, 0.00233)
	(0.39947, 0.24728)
sqrtSC	(-328.6298, 276.7716)
sqrtAI	(39.6513, 0.0000)
	(52.3985, 5.3925)
sqrtSC	(-104.2148, 0.0000)
sqrtNSAI	(27.2810, 29.1034)
sqrtSAI	(24.3339, 0.0000)
	(49.5368, 8.7168)

Table 4. Fuzzy regression model performance

Model	R^2	$adjR^2$
SC	0.880	0.870
CI		
sqrtSC	0.905	0.897
sqrtAI		
sqrtSC	0.902	0.889
sqrtNSAI		
sqrtSAI		

main characteristic of FLR analysis is to provide a fuzzy estimation model Y that can be regarded in different ways:

1. for any fixed x , the fuzzy model provide a range of possible values for y , meaning that estimation is not precise and should be assumed with a certain level of confidence;
2. Y groups a continuum of linear regression models: in SLR there is one model, and discrepancy between the data and the model are considered error; in FLR data points are considered error-free and several models are possible. This sounds reasonable in the context of SE metrics, where data is usually not affected by error, and a differences of data and functional dependency is only due to model approximation;
3. when, we fix y , there is a range of x within that estimation is possible with different degrees of possibility: there is one x value that gets possibility 1 in correspondence to y ; wandering from that value, leads to x value for which that estimation is less and less true, until reaching zero.

Therefore FLR analysis enriches the amount of information available for evaluation. However, this does not neces-

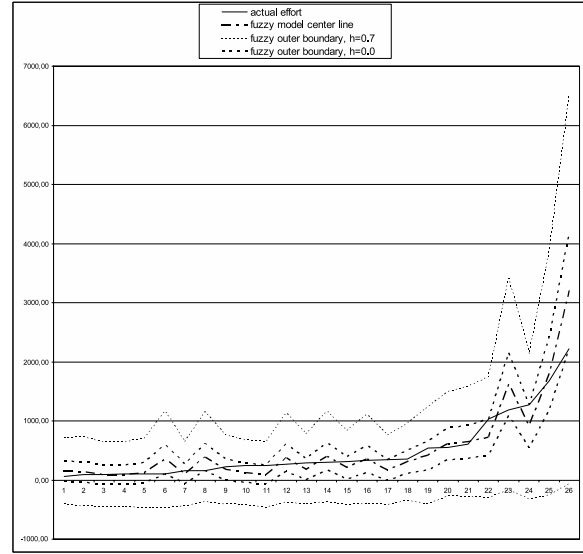


Figure 1. Fuzzy regression model (SC, CI)

sarily mean that the perception of the model is made simpler. One advantage of SLR is that the model is very simple and provides an immediate comprehension of phenomena, in exchange of model fitness. Differently FLR seems to provide more information and a higher fitness, but with a comprehension of phenomenon that could provide more difficult to the decision maker. More investigation should be made in order to better understand cognitive implications.

5 Conclusions and future works

In this paper we reported an initial experience of fuzzy linear regression aimed at estimating effort in software projects. In particular we adopted the possibility approach developed by Tanaka. The model provided a response that is comparable with a conventional statistical regression model, although assumptions are very different. Because of the fact that usually data points in Software Engineering are not affected by error, and uncertainty derives from suitability of a simplified model to available data, SLR and FLR approaches show different meaning and interpretation. The paper provided a brief qualitative comparison of both. At the moment no evident conclusion can be gathered. The initial result encourages a more deeper investigation of the usage of FLR analysis in SE in order to better understand application benefits and limitations. In particular, it would be useful to understand how differences in model can affect the evaluation of dependent variables in the context of SE, especially when statistical assumptions are violated.

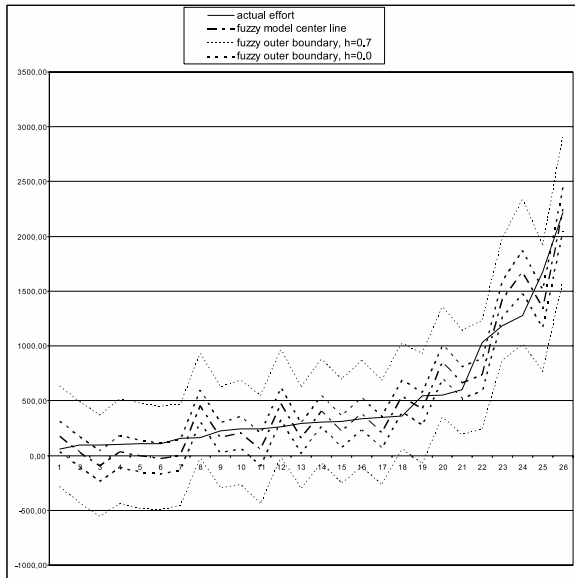


Figure 2. Fuzzy regression model (sqrtSC, sqrtAI)

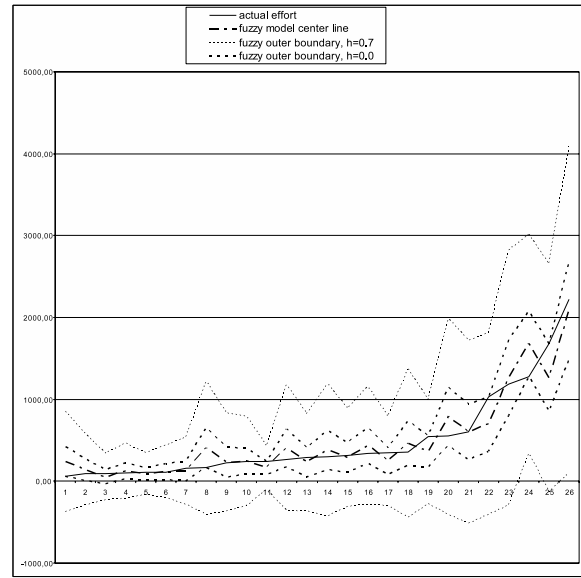


Figure 3. Fuzzy regression model (sqrtSC, sqrtNSAI, sqrtSAI)

6 Acknowledgements

We would like to thank Silvio Stefanucci for the help he gave us in interpreting the data of the case study [12].

References

- [1] A. Bardossy, I. Bogardi, and W. Kelly. Fuzzy regression for electrical resistivity - hydraulic conductivity relationships. In *North American Fuzzy Information Processing Society Workshop*, pages 333–352, 1987.
- [2] A. Celmins. Least squares model fitting to fuzzy vector data. *Fuzzy Sets and Systems*, 22(3):245–269, 1987.
- [3] P. Chang and E. Lee. Fuzzy linear regression with spreads unrestricted in sign. *Comput. Math. Appl.*, 28(4):61–70, 1994.
- [4] Y.-H. Chang and B. Ayyub. Reliability analysis in fuzzy regression. In *Proc. Annual Conf. of the North American Fuzzy Information Society*, pages 93–97, Allentown, PA, USA, 1993. NAFIPS.
- [5] Y.-H. O. Chang and B. M. Ayyub. Fuzzy regression methods – a comparative assessment. *Fuzzy Sets and Systems*, 119(2):187–203, 2001.
- [6] P. Diamond. Fuzzy least squares. *Inf. Sci.*, 46(3):141–157, 1988.
- [7] N. Fenton, P. Krause, and M. Neil. Software measurement: Uncertainty and causal modeling. *IEEE Software*, 19(4):116–122, /2002.
- [8] M. Gharpuray, H. Tanaka, L. Fan, and F. Lai. Fuzzy linear regression analysis of cellulose hydrolysis. *Chemical Engineering Communications*, 41:299–314, 1986.
- [9] H. Ishibuchi. Fuzzy regression analysis. *Fuzzy Theory and Systems*, 4:137–148, 1992.
- [10] K. J. Kima, H. Moskowitzb, and M. Koksalc. Fuzzy versus statistical linear regression. *European Journal of Operational Research*, 92(2):417–434, 1996.
- [11] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic; Theory and Applications*. Prentice Hall, Upper Saddle River, N. Y., 1995.
- [12] A. D. Lucia, E. Pompella, and S. Stefanucci. Assessing the maintenance processes of a software organization: an empirical analysis of a large industrial project. *The Journal of Systems and Software*, 65(2):87–103, Feb. 2003.
- [13] D. A. Savic and W. Pedrycz. Evaluation of fuzzy linear regression models. *Fuzzy Sets and Systems*, 39(1):51–63, 1991.
- [14] H. Tanaka and H. Ishibuchi. Identification of possibilistic linear systems by quadratic membership functions of fuzzy parameters. *Fuzzy Sets and Systems*, 41(2):145–160, 1991.
- [15] H. Tanaka, H. Ishibuchi, and S. Yoshikawa. Exponential possibility regression analysis. *Fuzzy Sets and Systems*, 69(3):305–318, 1995.
- [16] H. Tanaka, S. Uegima, and K. Asai. Linear regression analysis with fuzzy model. *IEEE Transaction Systems Man Cybernet*, 12:903–907, 1982.
- [17] S. Weisberg. *Applied linear regression*. Wiley, New York, 1985.
- [18] Z. Xu, T. M. Khoshgoftaar, and E. B. Allen. Prediction of software faults using fuzzy nonlinear regression modeling. In *Fifth IEEE International Symposium on High-Assurance Systems Engineering*, Albuquerque, New Mexico 2000.
- [19] L. Zadeh. The concept of a linguistic variable and its application to approximate reasoning-1. *Information Sciences*, 8:199–249, 1975.

An Intensional Tool Applied to French Language Educational Software

Honglian (Elena) Li, William W. Wadge
Department of Computer Science
University of Victoria, P.O. Box 3055
Victoria, BC, Canada V8W 3P6
elenali@uvic.ca, wwadge@csr.uvic.ca

Abstract. We present a Web-based French language educational software tool, French e-Flash Card (FFC), authored using our Intensional Markup Language (IML). The FFC site is interactive and dynamic and incorporates AI for French language grammar. These special features, lacking in most language educational tools (which are typically language-independent), are made possible by IML, which allows rule based markup in an intensional (possible worlds) logic. The IML source files, relatively small, define whole families of pages indexed by parameters which specify (for example) the particular subject matter, the degree of difficulty, even the choice of vocabulary and agreement constraints on French example sentences generated on demand. We describe the FFC site, give an overview of IML, and describe our design methodology, which should be applicable to many similar projects.

1. Introduction

This article presents three related topics: the French language educational software known as French e-Flash Card (FFC), the design approach used in creating FFC, and the authoring tool employed, Intensional Markup Language.

The French e-Flash Card was designed and implemented by the authors. It is based in part on Wadge and Wadge's original French Sentence Maker program. [1] The French Sentence Maker was programmed directly in ISE, a low-level Perl-like language with intensional features. FFC is a more ambitious project made possible by IML, which is not a programming language but rather a high-level markup language. IML is a rule-based extension of HTML which allows large chunks of ISE to be replaced by simple macro calls. It makes it much easier to produce the source for parameterized sentences and for tables, popup menus, search forms and the like.

FFC is in some ways typical logic based web application; it has a knowledge base, a logic engine, user selectable input parameters and a presentation layer for answers. It is an intelligent way to present the logical relations and rules underlying French grammar. It goes beyond most conventional Web based software in that it is not just a visual interface to a database. Since the FFC rules embody knowledge of French grammar, it is capable of generating student questions and example utterances that have not been explicitly entered in a database. Furthermore, the intensional logic behind IML supports an inheritance relation between contexts. This makes it easy to express the rules in terms of general principles and exceptions (and exceptions to exceptions, and so on). We provide a detailed description of the usage of the tool. We also investigate the difference between IML and the conventional web authoring tools.

We present the problem and our motivation in section 2; the design approach is explicated in section 3; we describe the use of intensional programming tool—IML in section 4; intensional logic and intensional programming are defined in the section; we illustrate the implementation of FFC in section 5; we conclude our work in the last section.

2. Problem Definition

Current French class-based educational software mainly sees French grammar knowledge as built up from small, simple facts; for example, the form a particular adjective takes in the feminine plural, or conjugated form of a particular verb in a particular person, number and tense. Students are typically taught and tested on their knowledge of these atomic facts. However, even simple French expressions use a surprisingly large number of these atomic facts. Therefore, beginning students of French often get frustrated when they attempt to use what they are learning in class. Some students can't even make well-formed sentences in a simple conversation.

Students need training in the basic facts of grammar but this knowledge alone in practice is not enough to allow them to easily produce complete sentences. They need to see plenty of examples of complete sentences.

There are quite a few French grammar websites or applications available nowadays. In reviewing them, we found that most of them still employ the class-based educational model. They usually move the contents of the French grammar textbooks online and include some exercises that are also based on the questions in particular grammatical units. These websites are simply electronic version of the grammar textbook. They are not really practical for students who try to use their learning.

These electronic textbooks have the same problem as their paper based counterparts: they provide only a limited supply of examples of complete sentences, namely the ones explicitly written, one by one, by the authors. There may be hundreds of them, but a student who spends even one week in France, immersed in French, will hear or read thousands and thousands. The electronic versions of the textbooks often have AI which automates the production atomic grammatical facts, but not that of complete utterances or even complete phrases.

FFC is a modest first effort to remedy this deficiency. The (still relatively simple) AI in FFC automates the production of complete phrases and sentences, as well as that of atomic facts. French e-Flash Card tries to provide as many sentences as possible in each grammatical unit, in order to present typical uses of the grammar facts. For example, in the possessive adjective section, the author built a set of examples that cover all possible uses of possessive adjectives: singular, plural, masculine and feminine. Of special importance is the fact that the student can guide the generation of the example sentences by choosing the vocabulary employed, so that, for example, she can see the difference between saying (in French) “the house is big”, “the car is big”, “the houses are big”, “the cars are big”, and so on. The student will discover, for example, that the French word for “big” has a different form in each of these four examples.

In designing and implementing this sentence-making function, it is necessary to solve the following technical problems:

1. How to make sure the user who has only limited knowledge of French grammar and vocabulary can make not only grammatically correct but also semantically meaningful sentences.
2. How to improve the customizability of the example sentences

3. How to provide dynamic examples to meet the user’s specific requirement and improve interaction between the user and the software
4. How to arrange the contents in a web interface in order to contain such a large quantity of example sentences.

3. FFC Design Approach

FFC offers the student a choice for each of the major elements of a sentence, such as subjects, predicates and objects. The students choose appropriate vocabulary items that they want to use in the sentence, without having to know the exact form and position these items will take in the complete sentence. Then FFC itself will handle the grammar rules interwoven in the sentence and generates the sentence for the user. The students can choose different vocabulary items or move to different grammatical sections to repeat the sentence-making process. As the result, the student can learn grammar by making sentences whose meaning he or she already knows but whose exact form he or she may not yet be able to produce.

The FFC French Grammar Knowledge Base handles not only the rules for atomic facts but also those (e.g. for subject-verb agreement or word order) needed to combine the atoms into a complete sentence. FFC can therefore answer questions of the type “but how do you say ...” which normally require the presence of a native speaker.

In order to improve the customizability and interactivity, FFC uses the parametric method to provide the user with a customized version of web page. These sentence elements are listed in a table or in a menu. It allows the user to customize his or her sentences with appropriate vocabulary items. Each option in a menu or each item in a table is an intensional link which consists of a version dimension and its version value. Once it is selected by clicking the mouse, the current version expressions will be changed. These different customized versions of the pages are not pre-stored on the server side; instead, they are generated on demand by the same sort of parametric (intensional) logic that the IML implementation uses to generate the French sentences. Figure 1 shows the architecture of the FFC.

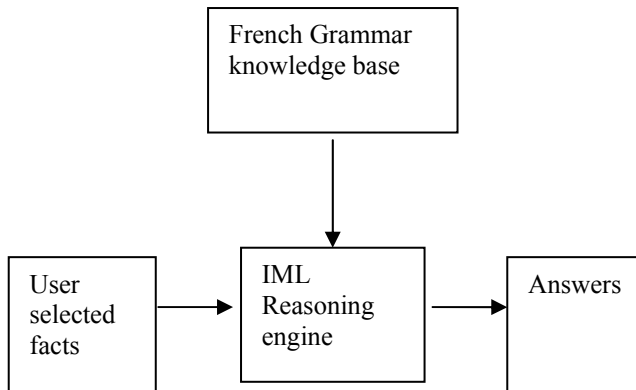


Figure 1. FFC is a logic based application

4. The Intensional Tool

Intensional Markup Language is the front end authoring tool to implement FFC. It is one of the intensional programming languages which are developed on the base of intensional logic by Wadge [2].

4.1. Intensional Logic

Intensional logic is the underpinning of intensional programming. “Intensional logic is therefore the logic of expressions in which the intension of sub-expressions (and not just extensions) has to be taken into account.” [3] It is very common in natural language. For instance, we can’t judge “It is raining” true or false unless we know the context such as when and where. Entities which vary according to context are called *intensions*, and each particular value determined by a particular context is called an *extension*.

Contexts are also called possible worlds, attributed to Leibniz by Chellas [4], Honderich [5] and others. Many variants of intensional logic have been described using possible worlds semantics, such as assignment of truth or false to a statement. For FFC, the set of possible worlds is a set of possible versions a sentence, or a whole page, or a whole site. In the simplest case, a version is a set of values for parameters. The logic used by FFC has a refinement relation. The fact that A \leq B means that the version B is a refinement of version A, which in turn (in the simplest case) means that B is consistent with A. But B assigns values to extra parameters not specified by A. In the logic used by IML, the refinement relation is a reverse inheritance relation. This means that version B of a page defaults to version A of that page, unless there the author provides explicit information about version B which overrides this default.

4.2. Intensional Programming

Intensional programming is programming using intensional logic. The obvious characteristics of intensional programming are using intensional operators to manipulate different versions.

The first intensional programming language is Lucid invented by Wadge and E. A. Ashcroft in 1974. [3] Its possible worlds if are a programmer-defined multidimensional space, where each dimension is defined as a non-negative integer. It has several operators, allowing access to the value at the next index in a dimension, the previous index, or all indices meeting some criteria. There is no direct connection between Lucid and IML. In particular, Lucid did not have a context refinement relation. However they share the same fundamental system which is demand-driven multidimensional processing.

4.3. Intensional Markup Language (IML)

The backbone of IML is ISE, a Perl-like CGI language with run-time parameterization. [2]. An IML package is a collection of Groff macro definitions. It conceals the complexity of ISE from the general web authors by using Groff macro definitions instead of the intensional expressions of ISE. [6] When all these tags are replaced by their definitions, the result is an ISE program.

It is ISE which implements the version space and refinement operation. In particular, ISE has a special case statement (the **vswitch**) whose alternatives are labelled with versions. On execution it chooses the alternative whose label is closest (in terms of refinement) to the current context. The **vswitch** with its *best-fit* principle is the main feature which implements the default/override logic vital to the construction of both example sentences and customized pages.

IML itself is a simple markup language that extends HTML. It is constructed on top of ISE. [6] The IML source is translated (once) into a corresponding ISE program which, when run with specific parameter settings, produces the HTML which renders to the desired version of the requested page. [2]

IML utilizes the parametric approach. It makes it possible for many members of a website community to share the pages by sharing parameters. Furthermore, it makes customization of web pages possible by altering the parameters. For instance, the user visits noun section of FFC, then the URL looks like the follows.

<http://i.csc.uvic.ca/elenali/FFC/new/ffc.ise<top:noun>>

The former part is regular URL. The ending “<top:noun>” is the parameter expression which present the user’s specific request for the noun section of FFC.

We use the IML “Autolink” construct to allow the student to add or delete parameters or to change their current values. As described before, different versions of a web page family are presented and invoked with different version parameters. Therefore, autolinks can be used for inter-version swap. The format of an autolink is:

```
.balink version_dimension:version_value
anchor_name
.ealink
```

The phrase in italic typeface is user-definable. An automatic link consists of a pair of tags “.balink” (begin a link) and “.ealink” (end a link) as well as the anchor name. A version expression, which consists of a version parameter and its desired value, follows the tag “.balink”. Once the user clicks the autolink anchor, the version expression is sent with the original URL to the web server reasoning engine. Then the best-fit version of the web page is calculated and sent to the user.

The FFC knowledge base is represented mainly using the IML “.gmod” and “.gcase” macros, which allow the author to specify rules for altering or adding parameter

values in the current context. Naturally, the rules are interpreted using best-fit logic.

```
.bgmod
.gcase current_version:value set_version:value
.egmod
```

If the current version dimension is null, then the set version dimension will be the vanilla version expression. Also, multiple gcase phrases can be nested.

The first rule specifies the default gender as masculine, while the other three override this default for the words *salade*, *chaise* and *voiture*. For example, when the user selects the noun “salade” by clicking the link which changes the current version as “nom:salade”, this block of code returns the gender version expression as “gen:e”, which encodes the fact that “salade” is a feminine word in French. Figure 2 is the implementation screenshot..

```
.bgmod
.gcase "" gen:-
.gcase nom:salade gen:e
.gcase nom:chaise gen:e
.gcase nom:voiture gen:e
.egmod
```

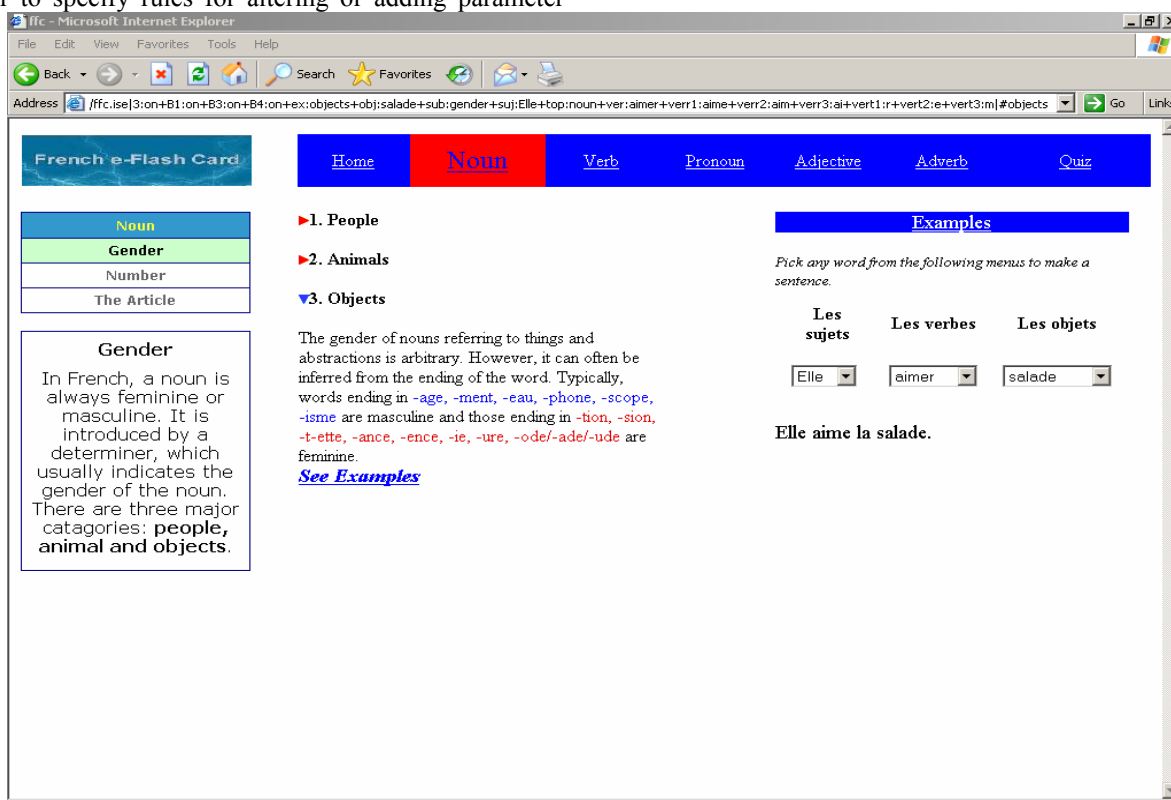


Figure 2. Screenshot of the gender of noun section in FFC

5. FFC Implementation

FFC is a typical application of intensional programming. The FFC site is a family of pages with thousands of versions, like the Noun section which has gender, number and article sub-sections. Each of these grammatical sections is defined as one of versions of web pages of Noun section. Although, their grammatical parameters encode information such as gender, number and article, they share same source of graphics and other attributes of a page such as top menus or footnotes. Therefore, the entire web site can be considered as versions of a single web page and related members can share source. The reasoning engine, as a version control system, allows code sharing between versions.

In FFC each composition of these elements of sentences is in fact a specific version of the web page family. FFC presents new examples as a new version of the web page on the user's demand. Therefore, FFC is a typical multiversion web application which has a concise layout and clear presentation.

5.1. Data Structure

The data structure of FFC is "tree" structure. There are a total of four levels of files. The file in the top level is `ffc.m`. The second level files include JavaScript menu files `topic_menu_array.js`, the layout files including `topicM.i` and `topicR.i`. The third level files are the example sentence generators for each subtopic, e.g. `enR.i` file is used to create examples for pronoun "en". The lowest level files are generic macrocodes, e.g. noun analysis macro definition file `announ.i`.

5.2. Functions

Number-alternative menu

The options in a series of menus can alternate between singular form and plural form. The change is decided by the current context. For example, the possessive adjective example generator employs the alternative menu. There are three menus: object menu, possessive adjective menu and the number menu, which controls the object in its plural or singular format. Any of these three menus can interact with the others. For instance, if the user chooses plural option from the third menu, the options in object menu and possessive adjective menu will change into plural form.

Vocabulary-alternative menu

The options in vocabulary-alternative menus can be changed into completely different vocabularies according to the previous selections of the user.

For example, the example sentence generator of `gender.m` in French noun topic employs the vocabulary-alternative menu. The user can choose verb and object from verb menu and object menu respectively. Each verb matches with a different set of objects. Once the user selects one verb, then the corresponding set of options for object will be presented in the object menu. What's more, the verb menu will present the corresponding conjugated verb according to the selection of subject.

Verb Conjugation Search Engine

FFC allows the user to look up the conjugation of the verb in present tense in Verb Conjugation Search Engine. The user can input his or her inquiries in the blank, and FFC takes the inquiry and searches for the conjugation form.

The search engine works in the following way: First, it catches the user's input and stores the string as a version dimension. Second, the verb analysis process is invoked to cut the string into single letters and store the last two to six letters into a word endings array, and the other letters into word a root array. Third, FFC will analyze the verb ending. It depends on the grammatical rules of French verb conjugation. Fourth, FFC goes through the irregular verb conjugation database to check whether the verb is irregular verb. The server side will use the best-fit algorithm to facilitate the search. If the verb falls into one of the irregular conjugation categories, it will be conjugated according to the specific rule. If the verb is just a regular verb, then the normal conjugation rule will be applied. Fifth, the conjugated verb ending will be added to the verb root in order to compose the conjugation verb. The last step is that FFC returns the result.

Let's see two examples. First, the regular verb "aimer" will be separated into two parts the verb root "aim" and the verb suffix "er". After checking through the irregular verb database, there is no record for "aimer". Therefore, it is automatically defined as a regular verb. The regular conjugation rule is applied to "aimer". Finally, FFC returns the result to the user.

The second example shows the irregular verb "manger". Verbs ending in -ger are irregular verbs which are defined in the irregular verb database. So FFC will conjugate the verb according to the specific rule and return the result to the user.

5.3. Comparison

Compared with current French language educational web applications, FFC breaks through the limitations of the traditional model of conventional language web applications, which is web interface plus database. In contrast, FFC is built on knowledge based model, containing French grammar knowledge base, logic reasoning engine and interface layer.

FFC has dynamic contents and layout. It has many more examples, generated from the different options selected by students themselves. Because the sentences change according to different selections chosen by the student, the student is rarely able to go through all the examples at one time. So the longer the student uses the application, the more contents and examples they can exploit from the application. It allows students to generate new sentences and examples, thus keeps their interest longer. Therefore, FFC promote an active learning curve.

FFC incorporates artificial intelligence. It employs all kinds of changeable menus, which can change the options according to the context. Moreover, it facilitates a semantic filtering out function. FFC is easy to maintain and be reused with simple and clear codes.

The methodology that we used to create this multiversion French educational application is an authoring formalism in which one can specify whole families of related pages with only modest additional effort to that required to specify a single page. After setting up the grammar rules for some cases, it can produce sentences after taking input from users. This is the artificial-intelligence incorporated in FFC, which is the distinctive characteristic of FFC.

6. Conclusion

The objective of this paper is to propose an innovative application of intensional programming in educational

language software. The objective is achieved through the development of the intensional multiversion web application French e-Flash Card. We presented FFC in terms of prototype design, intensional web engineering approach and implementation. In the process of describing the development of FFC, we also explicate the use of the intensional programming tool Intensional Markup Language. Moreover, the advantages of IML tool were discussed through the comparison with other intensional programming tools and conventional web authoring tools. What's more, we demonstrated the methodology of developing multidimensional web application through the developing of the application FFC.

References

- [1] W. W. Wadge and C. Wadge, "Multidimensional French", Proceeding of the Eleventh Annual International Symposium on Language for Intensional Programming, May 1998, Palo Alto, California, USA, W. W. Wadge, Ed., pp.109-117.
- [2] W. W. Wadge, "Intensional Markup Language," in Distributed Communities on the Web, Third International Workshop, DCW 2000, Quebec City, Canada, June 19-21, 2000, Proceedings, ser. Lecture Notes in Computer Science, P. G. Kropf, G. Babin, J. Plaice, and H. Unger, Eds., vol. 1830. Springer, 2000.
- [3] W. W. Wadge, Intensional Logic in Context. Intensional Programming II, World Scientific Publishing, Singapore, 2000, 1-13.
- [4] Brian F. Chellas. Model Logic: An Instruction. Cambridge University Press, 1980
- [5] Ted Honderich, editor. The Oxford Companion to Philosophy. Oxford University Press, 1995.
- [6] P. Swoboda, "Practical Languages for Intensional Programming," M. Sc. Thesis, University of Victoria, Canada (1999)
- [7] J. Girardet and J. Cridlig, Panorama de La Langue Française, ISBN 2.09.033709.5, CLE International, 2000

Analysis of meta-programs: a case study

Stan Jarzabek¹, Shen Ru¹, Hongyu Zhang² and Sun Zhenxin¹

¹School of Computing
National University of Singapore
Lower Kent Ridge Road, Singapore 117543
stan@comp.nus.edu.sg

²School of Computer Science and Information
Technology, RMIT University
Melbourne 3001, Australia
hongyu@cs.rmit.edu.au

Abstract

Meta-programs are incomplete and adaptable programs that are instantiated to meet a range of different requirements. Meta-programs in XVCL facilitate reuse and are organized into a hierarchy of meta-components, called x-frames, from which the XVCL processor generates concrete, executable programs. To aid in understanding of x-frames, we developed an x-frame query language, FQL for short. In FQL, we can formulate questions about x-frame properties. A FQL query processor automatically answers queries. An important finding from our study is that traditional static program analysis techniques are not directly useful for meta-programs. Interesting and useful queries require partial processing (that is instantiation) of a meta-program. While the details of analysis methods depend on a specific meta-programming technique, we believe readers interested in meta-programming techniques in general will find lessons from our experiment interesting and useful.

1. Introduction

Meta-programs represent a class of programs in generic form. Customized programs derived from the meta-program may differ in requirements, design decisions or platforms. Meta-programming techniques [2] facilitate reuse - customized programs form a product line, with a meta-program being a product line architecture.

Generic meta-programs are more difficult to describe and understand than concrete programs. Static analysis methods have been applied to ease understanding and maintenance of programs - can similar methods be used for meta-programs?

An important finding from our study is that traditional static program analysis techniques are not directly useful for meta-programs. Interesting and useful queries require partial processing (that is instantiation) of a meta-program. In this paper, we describe problems in understanding meta-programs built with XVCL [4][5] and a query system, our solution to some of those problems. XVCL, XML-based Variant Configuration Language, is a meta-language, method and tool for enhanced maintainability and

reusability developed at the National University of Singapore <http://fxvcl.sourceforge.net>. While the details of analysis methods depend on a specific meta-programming technique, readers interested in meta-programming in general may find lessons from our experiment interesting and useful.

2. An overview of XVCL

XVCL is based on Frame technology [1]. Frame technology has been extensively applied in industry to manage variants and evolve multi-million-line, COBOL-based, information systems. The excellent record of frame technology in large-scale software applications was the main reason which led us to implementing XVCL. Unlike original frames, XVCL blends with contemporary programming paradigms and complements other design techniques.

XVCL works on the principle of constructing custom systems by composing generic, reusable meta-components, after possible adaptations. Any location or structure in a meta-component can be a designated variation point, available for adaptation by ancestor meta-components. This “composition with adaptation” process turns meta-components into concrete components of the custom system we wish to build. Program generation rules are 100% transparent to a programmer, who can fine-tune and re-generate code without losing prior customizations..

To facilitate effective reuse, we split a program into generic, reusable and adaptable meta-components, called x-frames. An x-frame can be viewed as a component parameterized for change and reuse. Usually, from a small number of x-frames we can generate many concrete components that differ in various characteristics such as functional requirements or design decisions.

XVCL commands allow the composition of the meta-components, selection of pre-defined options based on certain conditions, etc. Meta-variables and expressions provide a powerful parameterization mechanism.

We organize x-frames into a layered meta-component architecture called an x-framework (Figure 1). An x-framework is designed into layers to enhance reuse. It is also “normalized” to eliminate redundancies.

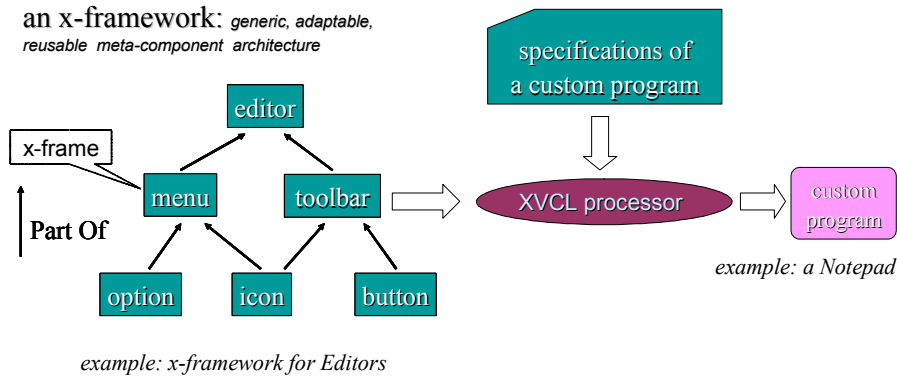


Figure 1. XVCL at work

XVCL is supported by a processor. The processor traverses x-framework, interprets XVCL commands embedded in visited x-frames and emits a custom program into one or more files. In our example, the output is emitted to a single file. The processor's traversal order is dictated by `<adapt>` commands embedded in x-frames. The `<adapt>` command tells the processor to customize and include the specified x-frame. This customization process of the x-framework is directed by instructions contained in the specification x-frame, called SPC for short.

2.1 Essential XVCL commands

We shall now describe a subset of XVCL that is sufficient for the purpose of this paper. We refer the reader to XVCL Web site for complete description of XVCL.

<x-frame> command:

```
<x-frame name= "name" >
```

x-frame body: mixture of code and XVCL commands

```
</x-frame>
```

An x-frame body contains the program code instrumented for ease of changing with XVCL commands. Attribute **name** defines the name of x-frame.

<adapt> command:

```
<adapt x-frame="name">
```

adapt-body : mixture of <insert>, <insert-before>, <insert-after> commands

```
</adapt>
```

(or)

```
<adapt x-frame="name"/>
```

The `<adapt>` command instructs the processor to:

- adapt the x-frame defined in "x-frame" attribute and apply the commands listed in the adapt-body,
- include the adapted x-frame into the current x-frame,
- resume processing of the current x-frame.

The x-frame **name** may be a character string specifying the name of a concrete x-frame (e.g., "A") or a reference to a variable whose value specifies x-frame name. Variable reference has a form "@V" (explained later in more details).

The adapt-body may contain a mixture of `<insert>`, `<insert-before>` and `<insert-after>` command. Customization commands may affect any x-frame reached in a sequence of `<adapt>` commands from a given one. For example, in x-framework of Figure 1, customization commands specified in `<adapt x-frame="editor">` may refer not only to x-frame *editor*, but also to x-frames *menu* and *toolbar*.

<break> command:

```
<break name="break-name">
```

break-body

```
</break>
```

The `<break>` command marks a point at which an x-frame can be adapted by ancestor x-frames via `<insert>`, `<insert-before>` and `<insert-after>` commands. The **break-name** is either character string or variable reference. The break-body defines the default code that may be replaced or extended by `<insert>`, `<insert-before>` and `<insert-after>` commands.

<insert> command:

```
<insert break="break-name">
```

insert-body

```
</insert>
```

The `<insert>` command replaces the break point "break-name" in the adapted x-frame and its descendents with the insert-body. The `<insert-before>` command inserts the insert-body **before** the break point "break-name" in the adapted x-frame and its descendents. The `<insert-after>` command inserts the insert-body **after** the break point "break-name" in the adapted x-frame and its descendents.

The insert-body may contain a mixture of code and XVCL commands to replace or extend at matching break point defined in "break" attribute.

<set> command:

```
<set var="var-name" value="value" />
```

The `<set>` command is used to define a single-value variable. The `<set>` command assigns a **value** defined in **value** attribute to single-value variable **var-name** defined in

var attribute. Value is either a string or a reference to a variable.

<set-multi> command:

<set-multi var="var-name" value="value1, value2, ..." />

The **<set-multi>** command is used to define a multi-value variable. The **<set-multi>** command assigns multiple values (*value1, value2,...*) define in "value" attribute to a multi-value variable name *var-name* define in **var** attribute. *<value-of> command*

<value-of expr = "variable-ref" />

The value of the "*variable-ref*" is evaluated and the result is inserted in place of the **<value-of>** command. Variable reference can be direct or indirect such as:

"@v" – value of variable v (direct reference)

"@@v" – value-of(value-of(v)) (indirect reference)

"@...@v" – multi-level indirect reference

Variable scoping rules:

Variable scoping rules are the same for both single-value and multi-value variables. While many x-frames may include **<set>** commands for the same variable, only **<set>** commands from one x-frame take effect during each run of XVCL processor through x-frames. The **<set>** command(s) in the ancestor x-frame takes precedence over **<set>** commands in its descendent x-frames. That is, once x-frame X sets the value of variable v, **<set>** commands that define the same variable v in descendent x-frames (if any) visited by the processor will not take effect. However, the subsequent **<set>** commands in x-frame X can reset the value of variable v. Variable v becomes undefined as soon as the processor returns the processing to the parent x-frame that adapts x-frame X.

Variables become undefined as soon as the processing level rises above the x-frame that effectively set variable values. This makes it possible for other x-frames to set and use the same variables and prevents from the interference among variables used in two different sub-trees in the x-frame hierarchy.

The above scoping rule has important implication on reuse. Lower level x-frames must be generic so that they can be reused in many systems. Such x-frames define default values of variables in their respective **<set>** commands. However, ancestor x-frames often need to adapt lower level x-frames for reuse in different contexts. Some of adaptations are done by setting values of variables. Therefore, ancestor x-frames must have a power to override defaults defined in lower level x-frames. Variable scoping rules in XVCL reflect the above thinking.

Definition of <select> command

<select option = "var-name">

select-body: may contain options listed below

</select>

select-body:

<option-undefined> (optional)

option-body

</option-undefined>

<option value = "value"> (0 or more)

option-body

</option>

<otherwise> (optional)

option-body

</otherwise>

The **<select>** command selects from a set of options based on variable "*var-name*" as follows:

<option-undefined> is processed, if the variable "*var-name*" is undefined,

<option> is processed, if value of "*var-name*" matches **<option>**'s "value",

<otherwise> is processed, if none of the **<option>**'s "value" is matched.

The option-body may contain a mixture of textual content and XVCL commands.

Definition of <while> command:

<while using-items-in="multi-var">

while-body

</while>

The **<while>** command iterates over while-body using the values of **multi-var** defined in **using-items-in** attribute. The i'th iteration uses i'th value of the multi-valued variable "multi-var". Inside while-body, **multi-var** with the i'th value can be used as single-value variable. The while-body may contain a mixture of code and XVCL commands.

3. Questions about x-frameworks

To effectively work with x-frameworks we must:

- understand the chain of x-frame inclusions triggered by **<adapt>** commands from SPC or from any internal point of an x-framework;
- understand the impact of such inclusions on properties of the resulting custom program
- understand the data flow relations at meta-level

Here are examples of specific queries programmers ask when trying to understand an x-framework:

1. Select all the **<adapt>** commands from all x-frames
2. Select variables that are modified in x-frame X
3. Select x-frames that modify value of variable COLOR
4. Select x-frames that adapt x-frame X
5. Select XVCL select structures that adapt x-frame X
6. Select x-frames adapted directly or indirectly from XVCL command at line 10 in x-frame X
7. Select x-frames containing break point named BP
8. Select x-frames that adapt x-frame X with modification at break point BP
9. Select x-frames that set variable v before adapting x-frame X
10. Select x-frames that use variable v in some **<adapt>** command
11. Select x-frames that are adapted directly or indirectly from x-frame X and modify variable V, and make

selection (in XVCL <select> command) based on variable v

4. An x-framework query language (FQL)

Answering the above queries is difficult and error-prone, as one has to inspect multiple x-frames and emulate XVCL processing to find the result. To ease understanding of x-frameworks, we designed a query language, called FQL (x-frame query language), and a FQL query system to automatically answer queries.

Ad hoc approach to supporting queries won't work as the list of queries is endless. We applied concepts from an earlier project on static program analysis [3] to design an x-framework query system in a systematic way as follows:

1. we start by identifying an useful class of x-frame queries such as those we listed in section 3,

2. we create a conceptual information model of x-frames that is needed to answer the queries,
3. we define the X-frame Knowledge Base (XKB), a repository to store the x-frame information,
4. we design a front-end to parse the x-framework and to generate the XKB contents
5. we implement an interpreter to evaluate x-frame queries written in FQL,
6. finally, we design x-frame view projector to display the results of x-frame queries.

Figure 2 depicts a conceptual model of XVCL commands described in section 2.

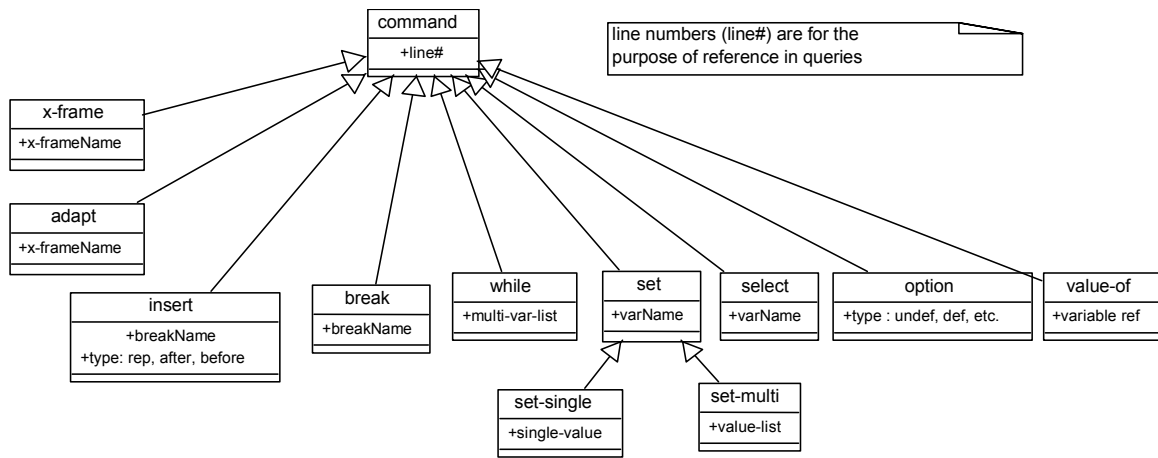


Figure 2. XVCL command model

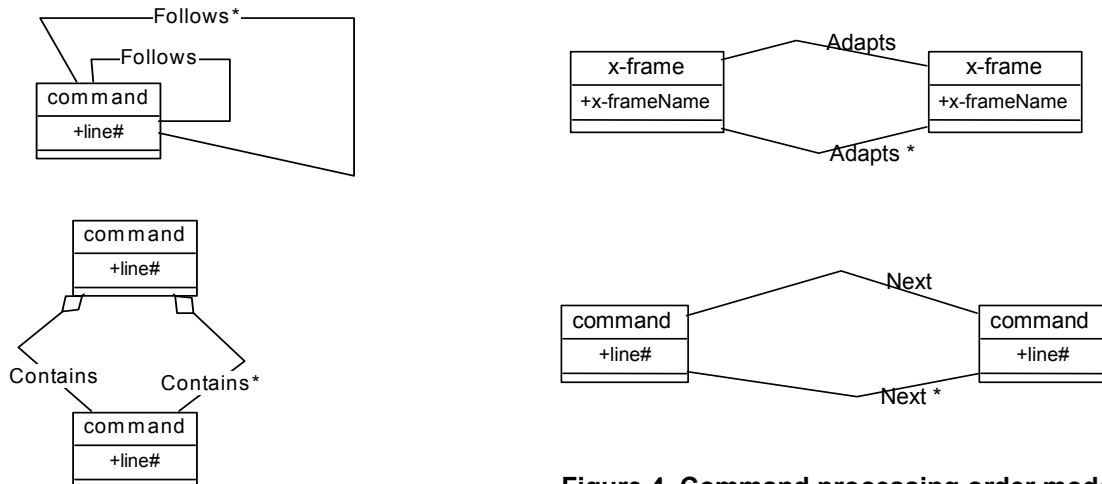


Figure 3. XVCL command structure model

Figure 4. Command processing order model

The following relationships among XVCL commands are of interest for querying purpose:

Relationship *Follows* (Figure 3) depicts the sequential order of commands in an x-frame. Relationship *Follows** is a transitive closure of *Follows*. Relationship *Contains*

models direct nesting among commands. Relationship *Contains** models direct or indirect nesting among commands.

Relationship *Adapts* (x, y) (Figure 4) holds if x-frame x adapts x-frame y in a processing context under consideration. Notice that x-frame x may contain a command `<adapt x-frame = "y">` but still this command may not be executed. Relationship *Adapts** is transitive closure of *Adapt*. Relationships *Next* and *Next** model processing control flow processing among any XVCL commands: *Next* (c, d) holds if commands c and d are in the same x-frame and command d is executed after command c. Relationship *Next** (c, d,) holds if command d is executed after command c, independently of whether commands c and d are in the same or in different x-frames.

Relationship *Modifies* (x, v) (Figure 5) holds if command x directly sets the value of variable v. Relationship *Modifies**(x, v) holds if *Modifies* (x, v) or there exist y such that *Adapts* *(x,y) and *Modifies* (y, v). Relationships *Uses* and *Uses** are defined in a similar way.

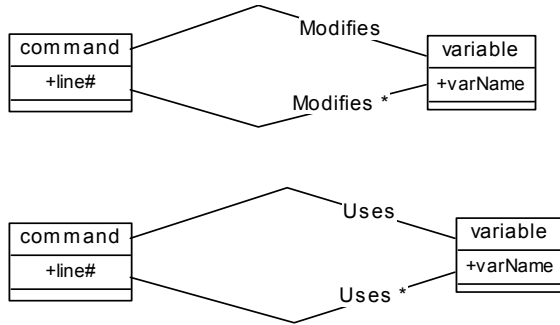


Figure 5. XVCL variable usage model

Relationships *Affects* and *Affects** (Figure 6) model data flow among variable definition and reference points. Relationship *Affects* (s, ref) holds if value of variable v assigned in `<set>` command s can be actually used in reference to this variable ref. Relationship *Affects** (s, ref) holds if there is a chain of variable definitions and references, starting at `<set>` command s and ending at variable reference ref, such that s affects (directly or indirectly) ref.

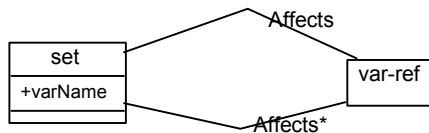


Figure 6. Data flow model

4.1 Querying x-frameworks in FQL

FQL queries are expressed in terms of x-frame information model described in the previous section. A typical query has the following format:

Select ... from ... such that ... with ...

Here are examples of queries:

Q1. Select all the adapt commands from all x-frames

Select adapt

Q2. Select all the `<adapt>` commands from x-frame “CAD”
x-frame x ; adapt a ;

Select adapt **from** x **with** x.x-frameName=“CAD”

or simply: **Select** a **from** “CAD”

Q3. Select all the `<insert>` commands directly nested within
`<adapt>` commands

Select insert **such that** Contains (adapt, insert)

Q4. Select variables whose values are modified in x-frame
X and used in x-frame Y

variable v ; x-frame x, y ;

Select v **such that** Modifies (x, v) **and** Uses (y, v) **with**
x.x-frameName = “X” **and** y.x-frameName = “Y”

Q5. Select x-frames that modify value of variable
“COLOR”

Select x-frame **such that** Modifies (x-frame, “COLOR”)

Q6. Select x-frames that adapt x-frame X directly or
indirectly

Select x-frame **such that** Adapts* (x-frame, “X”)

Q7. Select all references to variable v affected by `<set>`
command at line number 20 in x-frame “CAD”

var-ref ref ;

Select ref **from** “CAD” **such that** Affects (“CAD”.20, ref)

5. Interpretation of queries

As XVCL is a dialect of XML, we started prototyping FQL query evaluator using an XML query language XQL [6] and a public domain XML parser, JAXP from Sun Inc. [7]. We used JAXP parser for parsing XVCL files and extracting information about x-framework. The extracted information included the x-frame structure, adaptation hierarchy of x-frames, breakpoint settings, etc. We stored the extracted information in the XML format. We translated queries written in FQL into equivalent queries written in XQL and then used the XQL query engine to evaluate queries.

In this prototype solution, we could address only queries that could be answered by simple search for XML tags representing XVCL commands. Our evaluator could answer queries related to syntactical structure of x-frames (e.g., modeled by relationships *Follows* and *Contains*) and queries that required searching for different types of XVCL commands in x-frames. However, many useful queries depend on values of variables and cannot be answered by simple search for XML tags. For example, consider query:

Select x-frame **such that** Adapts (x-frame, “X”)

To answer this query, it is not enough to search for x-frames containing command <adapt x-frame = "X"/>. We must also consider commands <adapt x-frame = "@V"/> and check possible values of variable V. For this, we have to interpret the x-framework.

Variables are heavily used to parameterize x-frameworks: x-frame names in <adapt> or break names in <insert> are most often expressed in terms of variable references. Also, <value-of> is commonly used to represent class and method names in generic way. Such parameterization is an important technique to achieve reuse.

Interpretation of an x-framework depends on the context. The context is defined by specification x-frame SPC. The SPC usually sets values to many variables. Most of the variables have also default values that are defined anywhere in the x-framework. Therefore, we allow a programmer to specify the context in which a query is to be evaluated:

1. The default context in which no user-defined SPC is provided.
2. The customized context in which a programmer provides a SPC x-frame. Our FQL system also provides a user interface that allows a programmer to override default values of variables to set up a context for query evaluation.

6. Conclusions

In this paper, we described an analysis technique for meta-programs created with XVCL. In XVCL, we partition programs into generic, adaptable meta-components called x-frames and organize x-frames into a hierarchical structure called x-framework. An x-framework is meta-program that forms a product line architecture. To aid in understanding an x-framework, we developed a query language, FQL (x-frame query language). In FQL, we formulate questions related to adaptations and compositions of x-frames. A FQL query processor can automatically answer a class of useful queries aiding in understanding of x-frameworks.

An important finding of our study is that traditional static program analysis techniques are not very much useful for meta-programs. Interesting and useful queries require partial processing (that is instantiation) of a meta-program. Our query evaluator is, therefore, integrated with XVCL processor. We re-designed the XVCL processor and provided an API to facilitate invocation of XVCL processing functions during query evaluation. In addition, we provided an interactive environment for a programmer to set up a context for query evaluation (e.g., to set up

values of undefined variables) and to view the intermediate results.

In this paper, we described problems in understanding meta-programs built with XVCL and a query system FQL, and our solution to some of those problems. While the details of analysis methods to much extent depend on a specific meta-programming technique, the approach described in the paper can be applied to any meta-programming technique in which meta-components are parameterized for changes and composed after adaptations to build a specific program.

Flexible manipulation of programs, central in software reuse, is the strength of meta-programming. We believe that methods and tools for understanding and debugging of meta-programs are essential for wider acceptance of meta-programming as an effective approach to software development.

Acknowledgments

This work was supported by NUS Research Grant R-252-000-178-112.

References

- [1] Bassett, P. 1997. *Framing software reuse - lessons from real world*, Yourdon Press, Prentice Hall
- [2] Czarnecki, K. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, MA, 2000
- [3] Jarzabek, S. 1998 "Design of Flexible Static Program Analyzers with PQL," *IEEE Transactions on Software Engineering*, March 1998, pp. 197-215
- [4] Jarzabek, S., Bassett, P., Zhang, H. and Zhang, W. "XVCL: XML-based Variant Configuration Language," *Proc. Int. Conf. on Software Engineering, ICSE'03*, May 2003, Portland, pp. 810-811
- [5] Wong, T.W., Jarzabek, S., Soe, M.S., Shen, R. and Zhang, H. 2001 "XML Implementation of Frame Processor," *Symposium on Software Reusability, SSR'01*, Toronto, Canada, May 2001, pp. 164-172
- [6] W3C 1998, XML Query Language (XQL) Retrieved Oct 10, 2000 from the World Wide Web <http://www.w3.org/TandS/QL/QL98/pp/xql.html>
- [7] SUN Java Technology and XML. Sun Microsystems, Inc. Retrieved July 23, 2000 from the World Wide Web: <http://java.sun.com/xml/>.

Architectural Reflection in Adaptive Systems

Francesca Arcelli, Claudia Raibulet, Francesco Tisato, Marzia Adorni
Università degli Studi di Milano-Bicocca,
DISCo - Dipartimento di Informatica, Sistemistica e Comunicazione,
Via Bicocca degli Arcimboldi, 8, I-20126 – Milan, Italy
{arcelli, raibulet, tisato, adorni}@disco.unimib.it

Abstract. Architectural reflection enables software systems to observe and control their own structure and behavior. This fact is particularly useful in enhanced distributed systems that should adapt dynamically to the mobility of users, the heterogeneity and ever changing access devices and networks, and/or other run-time modifications that may influence the access of applications (i.e., information and services). The architecture proposed by this paper introduces a reflective layer which aims at capturing those features of a system that are relevant at run-time for the overall system adaptability. These features are expressed in terms of quality of services. The paper focuses on the description of the reflective layer, including its position within the overall system architecture and its specification using the object-oriented paradigm.

1 Introduction

Enhanced distributed systems should enable various types of devices and networks to be dynamically integrated within their structure [8, 14]. And this is primarily because of the mobility of users [2, 3, 17] and the heterogeneity [2, 6, 8] of devices and networks through which applications are accessed.

To achieve this challenging issue, a system should be able to adapt itself: either to adapt applications [15, 17, 22] (i.e., information, services) to the system features (i.e., devices/network features), or to adapt the system features [2, 20] (whenever it is possible) to applications' requirements. However, adaptability [2, 17] is a complex argument being required at various levels (i.e., from physical networks to communication protocols, from physical resources to applications, etc.) and regarding a wide range of issues that are determinant at run-time (i.e., resource usage, location, costs, computational and communication performances, etc.).

There are various approaches [2, 15, 17, 20, 22] that aim at addressing adaptability. They may be divided in two categories: (1) rather ad-hoc solutions which address adaptability only at the application level, and (2)

enhanced solutions which address adaptability also at the architectural level. In this context, reflection [1, 12] seems to be particularly suitable to address adaptability in that it enables a system to inspect its internal structure and behavior. Architectural reflection [4, 20] represents explicitly architectural aspects of a system, and moreover, exploits these aspects.

Generally, architectural reflection introduces an additional layer which plays an intermediary role between the system representation and applications. This layer enables applications to adapt to system features, and vice-versa, systems to adapt (whenever it is possible) to applications' requirements. In this context, a reflective layer should capture only information that is related to the system representation and that is meaningful at run-time. This information consists of non-functional aspects. A reflective layer is causally connected to the logical layer, which models system components and functionalities.

This paper presents a reflective architecture for adaptive systems. The reflective layer defined by the architecture captures the quality of services (QoS) [5, 9, 19] of the system components. We consider that QoS are particularly relevant for an adaptive system in that its adaptability is partially (if not totally) determined by its capability to provide services which are characterized by various levels of QoS. We have extended the term QoS also to devices (computational components), hence we call QoS all device and network features that determine system performances at run-time. The reflective layer is specified in object-oriented terms.

In addition to its fundamental objective (of enabling the observation and control of a system components), the reflective layer aims at being as light as possible in order to avoid increasing significantly the number of software components and consequently, reducing overall efficiency. Further, the layer has been designed to be flexible enough to allow modifications of itself without causing overall damage.

The rest of this paper is organized as follows. Section 2 describes the reflective architecture of an adaptive system. The reflective layer is presented in Section 3. Section 4 introduces implementation considerations

regarding an adaptive system that uses our reflective architecture. Related work is discussed in Section 5. Conclusions and further work are dealt within Section 6.

2 Architectural Issues

The architectural model described in the following has been designed for a multi-channel adaptive system [13] enabling users to access information and services through various types of devices and networks. The architecture is characterized by the following layers (see Figure 1).

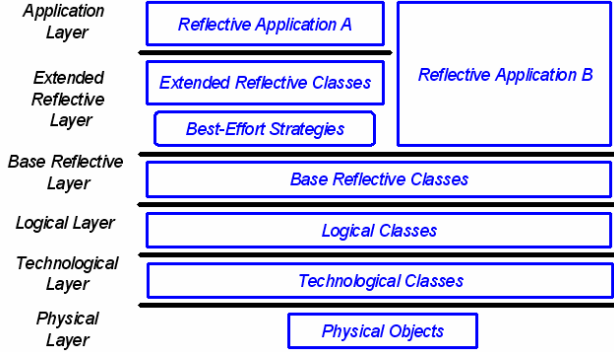


Figure 1 – The Architectural Layers

The *technological* layer provides the visibility of the system objects via platform-dependent mechanisms.

The *logical* layer represents an abstraction of the technological layer in that it enables the composition of physical components into logical aggregates.

The *base reflective* layer defines basic reflective classes allowing QoS features to be observed and controlled in a platform- and standard-independent way. According to the separation of concerns principle, such classes do not embed any policy.

The *extended reflective* layer embeds best-effort strategies which provide an improved level of QoS.

The *application* layer consists of domain specific applications. Figure 1 does not highlight that, in general, applications perform their job by exploiting domain-specific knowledge via functional, non-reflective features of the system objects. *Reflective applications* exploit both domain and reflective knowledge. They either rely on best-effort strategies (Appication A) or implement domain specific strategies on the base reflective layer (Appication B).

The base and extended reflective layers provide the same interface to the application layer even if their implementations do not coincide.

3 The Base Reflective Layer

The goal of the base reflective layer is to define basic, fundamental abstractions which capture information of an adaptive system in terms of QoS. This section presents the

core classes of the base reflective layer, which may be further specialized for actual systems.

In order to avoid confusions between non-reflective (i.e., classes specified by the technological and/or logical layers) and reflective classes (i.e., classes specified by the base and/or extended reflective layers) a naming convention has been adopted: “R_” (Reflective) prefix is used to denote reflective classes.

R_Object is the superclass of any reflective class. As shown in Figure 2, an instance of R_Object may be associated by a causal connection relationship to an instance of Object (the superclass of non-reflective system objects). In this way, R_Object reflects (being a meta-representation of) the Object. The specification of non-reflective classes is out of the scope of the reflective layer, which instead implements *causal connection* mechanisms between reflective and non-reflective information (see Section 3.4). Note that there may exist R_Objects that do not have a direct causal connection to Objects. (i.e., R_CompositeComponents which model R_Components aggregations - see Section 3.1).

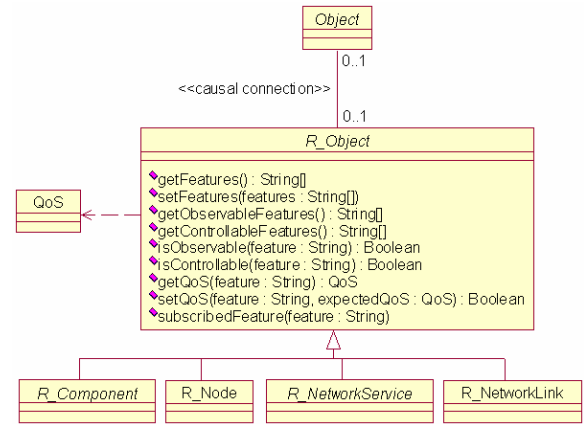


Figure 2 – The Core Reflective Classes

R_Object provides methods to discover and specify which are the QoS-related features for a specific object (getFeatures() and setFeatures()). Further, it provides methods to inspect which QoS features are visible (i.e., observable and controllable) for a specific object. getQoS() and setQoS() methods are used to inspect and define the QoS value provided by a specific feature. Concrete subclasses of R_Object are expected to implement getQoS() and setQoS() methods by exploiting the proper subclasses of QoS (see Figure 3).

QoS features are identified by symbolic names (Strings). For example, in a Java implementation of the base reflective layer, the introspection mechanism [21] can be exploited both to implement the inspection methods and to invoke them by symbolic names.

R_Object supports the publish-subscribe pattern [7] (i.e., an observer can subscribe a specific feature in order to be asynchronously notified about its change). This is

particularly important for adaptive applications that may exploit this mechanism to be notified about any modification in a system.

There are four main subclasses of `R_Object`. `R_Component` models computational components that are relevant for their QoS features. `R_Node` models network components through which `R_NetworkServices` may be reached. `R_NetworkLink` models the QoS of an actual connection between an `R_Node` and an `R_NetworkService`.

The QoS class (see Figure 3) used by `R_Object` is the superclass of any QoS feature. Due to the fact that QoS depend strongly on the application domain, it has been considered useful to define a general scheme that establishes how QoS should be defined, rather than providing a list of possible QoS features. In this way, the same component may exhibit different QoS features depending on the application domain.

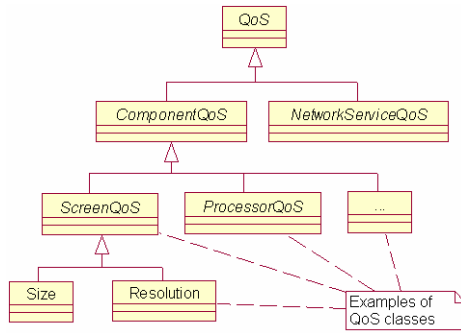


Figure 3 – The QoS UML Class Diagram

The diagram shown in Figure 3 indicates how QoS classes should be defined. Subclasses of QoS are types (i.e., classes whose attributes are of elementary types) that quantify the QoS of specific features. Note that QoS features are modeled as leaves in the hierarchical diagram. Hence, leaves can be added and/or removed without influencing other parts of the model. For instance, Size and Resolution are modeled as subclasses of ScreenQoS. The naming convention specifies that for example, Size/Resolution is the subclass of the QoS that quantifies the QoS feature “Size”/“Resolution”.

3.1 Computational Components

`R_Component` (see Figure 4) models the QoS of any computational entity of an adaptive system. Examples of `R_Components` are: PCs, PDAs, memory, processors, software components, etc. Note that `R_Component` may model something that does not correspond to an actual component at the technological layer, but to an aggregate at the logical layer. For example, an `R_CompositeComponent` may be a logical aggregation of two or more `R_ElementaryComponents`, hence it does not have a technological

equivalent. As it can be easily observed the model exploits the Composite Design Pattern [7].

An `R_Component` has associated a `Location`, which indicates the physical location of an `R_Object`. Location may be expressed in absolute, relative, geographical coordinates, etc. For instance, to locate the PCs in a building, the building plan may be used.

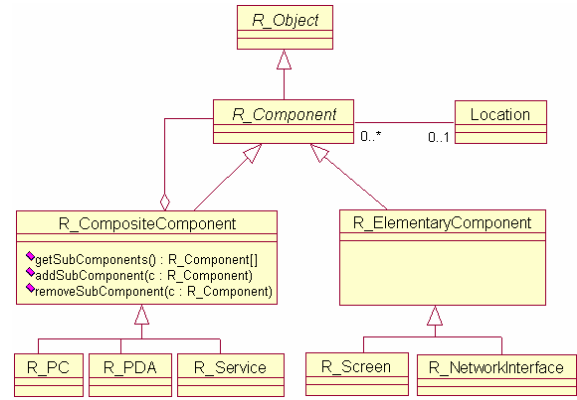


Figure 4 – The `R_Component` UML Class Diagram

An `R_ElementaryComponent` models a technological and/or logical object that can be effectively observed and/or controlled. It reflects the QoS of a piece of software and/or hardware (i.e., a peripheral and its related driver) which can be considered atomic from the QoS point of view (its internal structure is not meaningful). Examples of elementary components are: screen, keyboard, processor, router, network interface, etc. A particular example that may require further explanation is the `R_NetworkInterface`. This class models the components that enable the access to a (type of) network (i.e., net card, wireless card, etc.).

`R_CompositeComponent` models the QoS of a component which is meaningful from the QoS point of view, but which is not directly associated to a corresponding Object. The `R_CompositeComponent` is an aggregation of subcomponents (which can be both `R_ElementaryComponents` and `R_CompositeComponents`). Examples of composite components are: PC, PDA, application service, etc.. Note that a PC or a PDA may be seen as `R_ElementaryComponents`, too, if their structures are not meaningful for a particular application domain.

3.2 An Actual Example: `R_Screen`

Generally, a screen provides a visualization service. The QoS of this service are determined by screen's dimensions, resolution, color quality, etc.

An example of a Screen component is shown in Figure 5. Note that Screen and `R_Screen` are two different classes that are causally connected to each other. Screen is characterized by three attributes that express

the Size and Resolution of a screen. The class provides methods to get and set its attributes, and to display images. *R_Screen* describes a screen through two QoS features: Size (observable) and Resolution (observable and controllable). Size is translated into inches within the Screen class, while Resolution into horizontal and vertical resolution.

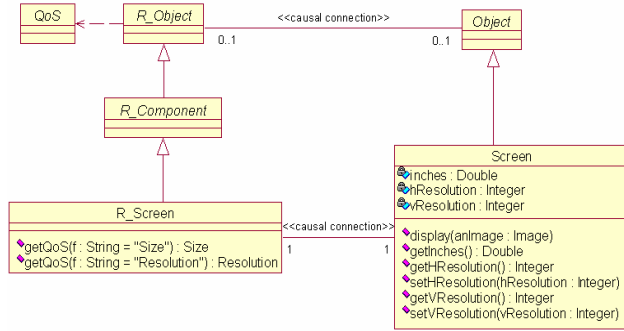


Figure 5 – R_Screen and Screen Classes

The behaviour of a reflective (adaptive) application that *observes* and *controls* the QoS of a Screen is described by the collaboration diagram shown in Figure 6. The application displays images on a screen. It observes the “Resolution” feature by invoking `getQoS(“Resolution”)`. After obtaining the “Resolution” value, the application verifies that the image may be displayed properly using the actual resolution. If the answer is no, the application requires the modification of the resolution to the *R_Object* (which automatically generates the modification of the resolution values in the Screen object by being causally connected to the last). Then, the application displays the image invoking the `display()` method. This is a typical case in which system features adapt to the application requirements.

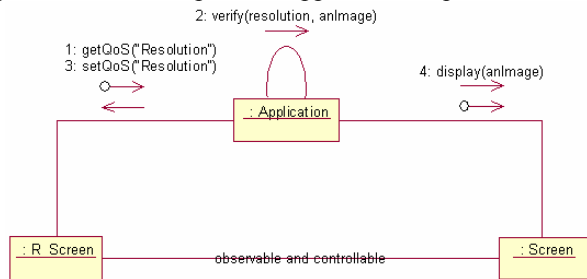


Figure 6 – The Collaboration Diagram for Observing and Controlling a QoS Feature

3.3 Network Components

An *R_NetworkComponent* (see Figure 7) is an *R_Component* that is actually connected in a network.

Conceptually, it makes the link between *R_Component* class diagram (which models

computational components) and *R_Node* class diagram (which models network components). For instance, a laptop connected to a network becomes an *R_Node*.

A network may consist of inner nodes (*R_Nodes*) and of end-nodes (that may either *R_Nodes* or *R_NetworkComponents*). We have considered that an *R_Node* is meaningful at the reflective layer in that it provides a network address through which *R_NetworkServices* may be accessed. The actual connection of an *R_Node* to an *R_NetworkService* is described by the association class *R_NetworkLink*, which provides all the QoS of the connection.

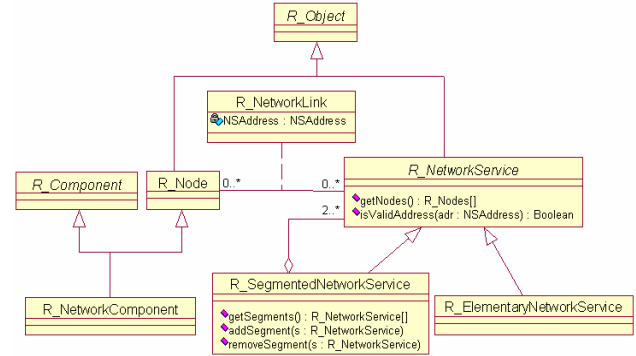


Figure 7 –The Network UML Class Diagram

As *R_Components*, also *R_NetworkServices* may be composed. Hence, the Composite Design Pattern has been used to describe *R_NetworkServices*. An *R_ElementaryNetworkService* is the equivalent of the *R_ElementaryComponent*, while the *R_SegmentedNetwork* corresponds to the *R_CompositeComponent*.

As the class names suggest, the approach of describing networks is top-down: having an *R_NetworkService*, we obtain improved QoS if we are able to segment it and consequently to observe and control its sub-components. While, in the case of *R_Components*, the approach is bottom-up: we obtain improved (computational) QoS if we aggregate simple components.

3.4 Causal Connection

To achieve causal connection between non-reflective and reflective objects, two strategies have been considered

Observation strategy uses the `update()` method, which aligns the information of the reflective layer to the information of the logical/technological layer. Implicitly, this method enables applications to observe the system features and to adapt themselves accordingly.

Control strategy uses the `force()` method, which aligns the information of the logical/technological layer to the information of the reflective layer. This method enables applications to control the system features and to

force components to adapt to application requirements. Note that this method is not always successful.

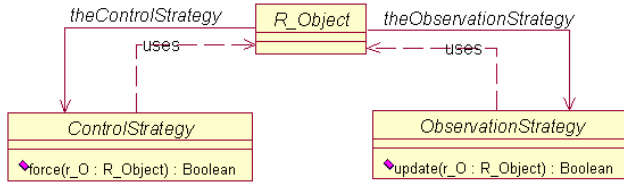


Figure 8 – Causal Connection Strategies

These two strategies are independent of any strategy defined at the extended reflective or application layer.

4 Implementation Issues

An actual multichannel adaptive system example is presented within Figure 9. Being designed as a decentralized system, the architecture of each device conforms to the model described within Section 2 (see Figure 1).

Currently, the initialization of technological, logical, and reflective objects are performed using a configuration file specified in XML terms. The network description is stored in a centralized repository.

Applications access and use both reflective objects and logical/technological objects of a device. Further, they interrogate the network repository to find out which are the network services (*R_NetworkServices*) a device may access via its *R_NetworkInterfaces*, and which are the available devices (*R_Components*) accessible through a specific network service. Applications are implemented in Java.

The application example considered for testing our architecture regards remote display of images. We have individuated two types of images. The first claims that the entire image should be displayed to maintain its significance (i.e., an image showing a person, pet, landscape). Hence, the application asks the remote device about its display QoS features. If there are any QoS features that are controllable, the application tries to force (if possible) the remote device to change accordingly to its desired QoS requirements (i.e., resolution, color quality, etc.). Then, the application itself tries to adapt the image to the remote device features. Finally, it sends the image to the remote device to be displayed.

The second case regards images in which a part of an image may substitute the entire one. For example, images that show colours or textures conserve their meaning even if not displayed entirely. For such images, the application asks the remote device about its display QoS features and sends a part or the entire image (if possible) to the remote device to be displayed.

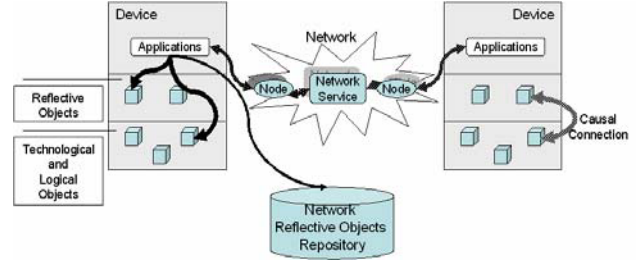


Figure 9 – An Example of a Multichannel System

Note that transmission of images to the remote device takes into consideration network QoS features. Hence, applications may adapt (i.e., compress) further images according to network QoS.

Further work on this example aims at enabling devices to discover network services and their related information through multicast/broadcast message, and without using a centralized repository. In this way we aim at implementing a totally decentralized multichannel adaptive system.

5 Related Work

Approaches that address adaptability through reflection are presented in [2, 3, 11, 15, 17, 20]. They are designed for specific types of systems or application domains such as mobile environments, Internet applications, multimedia, and multi-channel systems. The advantage of our approach is that it is independent of any specific application domain. It provides a set of general rules that can be further specialized and personalized for specific applications. Hence, we provide an adaptable solution that provides support for developing adaptive systems.

Examples of research projects that address adaptability especially in Internet, mobile-enabled environments, and multi-channel systems are: Odyssey [18], which defines a platform to manage adaptive applications for various mobile devices, Ninja [16], which defines a software infrastructure for Internet applications based on Web Services by providing composition, customization, and accessibility from a wide range of devices, Chisel [10], which defines an open framework for dynamic adaptation of services in a policy-driven, context-aware manner, and MAIS [13], which aims at providing methodologies, environments, and tools for developing multi-channel adaptive information systems.

6 Conclusions and Further Work

This paper has presented an approach to address adaptability through architectural reflection. The main goal of our solution has been to define abstractions that enable the observation and control of systems' features at run-time.

Our solution presents at least two advantages. First, separation of concerns is achieved. This aspect is related both to the separation of system objects from domain objects, and to the separation of the base reflective objects (defined at the base reflective layer) from strategies (defined at the extended reflective or application layer) that may exploit these objects. Second, the reflective layer ensures flexibility by separating reflective classes from strategies and by defining QoS externally to the reflective classes. In this way, modifications of strategies and/or QoS classes do not influence the reflective model.

Another interesting aspect of our approach is that it takes into consideration also QoS features related to computational elements, which may influence the performances of a system or an application. Obviously, computational elements provide only few QoS aspects which may change seldom with respect to the QoS related to the network. Network QoS have been only mentioned in this paper because similar solutions are available in other works [2, 17]. However, there are cases in which these aspects may influence significantly the overall performances of a system or application.

Part of this work has been performed within the research project MAIS¹ – “Multichannel Adaptive Information Systems” [13]. Further work consists in the refinement of the base reflective layer, the implementation of best-effort strategies at the extended reflective layer, and the implementation of other applications (especially disaster recovery) to test and validate our solution.

References

- [1] C. Bekker, P. Putter, “Reflective Architectures: Requirements for Future Distributed Environments”, Proceedings of the Fourth Workshop on Future Trends of Distributed Computing Systems, 1993, pp. 112-118.
- [2] G.S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran, N. Parlavantzas, K. Saikoski, “A Principled Approach to Supporting Adaptation in Distributed Mobile Environments”, Proceedings of the International Symposium on Software Engineering for Parallel and Distributed Systems, 2000, pp. 3-12.
- [3] L. Capra, “Mobile Computing Middleware for Context-Aware Applications”, Proceedings of the 24th International Conference on Software Engineering, 2002, pp.19-25.
- [4] W. Cazzola, A. Savigni, A. Sosio, F. Tisato, “Rule-Based Strategic Reflection: Observing and Modifying Behaviour at the Architectural Level”, 14th IEEE International Conference on Automated Software Engineering, 1999, pp. 263-266.
- [5] D. Chalmers, M. Sloman, “A Survey of Quality of Service in Mobile Computing Environments”, IEEE Communications Surveys, 1999, pp. 2-10.
- [6] V. Demesticha, J. Gergic, J. Kleindienst, M. Mast, L. Polymenakos, H. Schulz, L. Suredi, “Aspects of Design and Implementation of a Multi-channel and Multi-modal Information System”, Proceedings of the IEEE International Conference on Software Maintenance, 2001, pp. 312-319.
- [7] E. Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns: elements of reusable object-oriented software”, Addison Wesley, Reading MA, USA, 1994.
- [8] D. Garlan, “Software Architecture: a Roadmap”, Proceedings of the Conference on The Future of Software Engineering, 2000, pp. 91-101.
- [9] J. Gozdecki, A. Jajszczyk, R. Stankiewicz, “Quality of Services Terminology in IP Networks”, IEEE Communications Magazine, Vol. 41, No. 3, 2003, pp. 153-159.
- [10] J. Keeney, V. Cahill, “Chisel: A Policy-Driven, Context-Aware, Dynamic Adaptation Framework”, Proceedings of the Fourth IEEE International Workshop on Policies for Distributed Systems and Networks, 2003, pp. 3-14.
- [11] F. Kon, F. Costa, G. Blair, R.H. Champbell, “Adaptive Middleware: The Case for Reflective Middleware”, Communications of the ACM, Vol. 45, No. 6, 2002, pp. 33-38.
- [12] P. Maes, “Concepts and Experiments in Computational Reflection, Proceedings of the Object-Oriented Programming Systems, Languages and Applications (OOPSLA’87), 1987, pp. 147-155.
- [13] MAIS project - <http://www.mais-project.it/>
- [14] A. Maurino, S. Modalferi, B. Pernici, “Reflective Architectures for Adaptive Information Systems”, Proceedings of the Workshop on Multi-Channel and Mobile Information Systems, 2003.
- [15] P. Motuzenko, “Adaptive Domain Model: Dealing with Multiple Attributes of Self-Managing Distributed Object Systems”, Proceedings of the 1st International Symposium on Information and Communication Technologies, 2003, pp. 549-554.
- [16] Ninja - <http://www.gigascala.org/mescal/forum/17.html>.
- [17] B. Noble, “System Support for Mobile, Adaptive Applications”, IEEE Personal Communications, Vol.7, No. 1, 2000, pp. 44-49.
- [18] Odyssey - <http://www-2.cs.cmu.edu/afs/cs/project/coda/Web/docs-ody.html>.
- [19] QoS Cisco- http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm.
- [20] J. Suzuki, Y. Yamamoto, “OpenWebServer: An Adaptive Web Server Using Software Patterns”, IEEE Communications Magazine, Vol. 37, No. 4, 1999, pp.46-52.
- [21] Sybex, Inc.: Enterprise Java 2, J2EE 1.3 Complete, Sybex Inc, (2003)
- [22] J.W. Yoder, F. Balaguer, R. Johnson, “Architecture and Design of Adaptive Object-Models”, ACM SIGPLAN Notices, Vol. 36, No. 12, 2001, pp. 50-60.

¹ The MAIS research project is financed by MIUR – “Ministero dell’Istruzione, dell’Università e della Ricerca” in the context of the FIRB program – “Fondo per gli Investimenti della Ricerca di Base”.

Automated Assistance for Eliciting User Expectations

Orna Raz[†], Rebecca Buchheit[‡], Mary Shaw[†], Philip Koopman^{*}, Christos Faloutsos[†]

[†]School of Computer Science, [‡]Civil Engineering Department, ^{*}ECE Department
Carnegie Mellon University
Pittsburgh PA 15213 USA

E-mail:{orna.raz, rebecca.buchheit, mary.shaw, koopman, christos}@cmu.edu

Abstract

People often use software for mundane tasks and expect it to be dependable enough for their needs. Unfortunately, the incomplete and imprecise specifications of such everyday software inhibit many dependability enhancement techniques because these require a model of proper behavior for failure detection. We offer a user-centered approach for creating a model of proper behavior. This approach is based on satisfying the user expectations—software behavior the user relies on—rather than demanding perfect specifications. It utilizes data mining through a novel template mechanism, to help users make their expectations precise. The resulting precise expectations can then serve as proxies for missing specifications in detecting unexpected data behavior. We concentrate on data feeds: continuous streams of data, a challenging example of everyday software. Using our method on a real world data feed, it took just hours to detect problems that had taken the data providers months to detect independently. These problems surprised even our user—a domain expert that had previously analyzed the same data feed. Systematic analysis further supports the usefulness of our method.

1. Introduction

If all software had perfect specifications—precise, complete, and correct, increasing the dependability of everyday software elements would be straightforward: use the specifications as a model of proper behavior and detect a failure when the software’s behavior is outside the specifications.

Unfortunately, specifications are rarely, if ever, perfect. Moreover, it is neither cost-effective nor feasible to strive for perfect specifications for *everyday software elements*—elements incorporated in applications that are neither mission nor safety critical. Yet, the utility of such elements would greatly increase with increased dependability.

Data feeds are an example of everyday software elements and the one we use in our work. A data feed is a time ordered sequence of observations on output. Data feeds may remain under the control of their providers and may have many users relying, in different ways, on behavior the providers did not anticipate. Many challenging, real world software elements fall under the category of data feeds, in-

cluding Internet services and software elements that process sensor data or perform monitoring activities. Examples include quotes for a stock, weather forecasts, and the truck weigh-in-motion data we use in this paper.

We propose a user-centric approach for coping with incomplete specifications of data feeds. Our method helps users make their expectations about data feed behavior precise. It can then automatically detect *semantic anomalies*—data feed behavior that falsifies these expectations. It applies statistical and machine learning techniques to help discover meaningful information in the data. These techniques precisely characterize various aspects of the data. However, to characterize *relevant* behavior, our method must elicit the user expectations as well. It does so via a novel *template* mechanism. In essence, templates document the predicates of the inference techniques.

The template mechanism is the main contribution of this paper. The case study provides empirical evidence in support of its usefulness.

Each user relies on a data feed in a certain way and expects the behavior of the data feed to support this usage. Therefore, a given user may only care about a subset of the data feed properties. Moreover, a user may care about behavior that is missing from existing specifications or even unnoticed by the providers. However, users’ expectations are informal and imprecise, though they are reasonably accurate. For example, a user may expect trucks reported by an on road scale to be physically feasible but may not be able to specify all the properties and values that define such feasibility.

Our approach has the advantages of (1) requiring no knowledge about inputs or implementation details, including source code or binaries and (2) requiring no user data mining expertise. All it assumes is that (1) it can observe the data feed over time, as the user uses it, (2) this usage will tolerate recognition and repair of faults rather than require prevention, and (3) the user has enough domain knowledge to select predicates from a list our method automatically generates. We talk about anomalies rather than failures because our approach, like any dynamic analysis, is potentially unsound. However, our case study shows it can be highly useful in practice.

Our approach is domain independent. Encouragingly, it

was able to produce results that were interesting within the application domain of our case study: monitoring systems in civil engineering; domain specific details and results are described in a paper intended for civil engineers [21].

Our case study is a real world truck “weigh-in-motion” (WIM) system using a standard data feed from the Minnesota Department of Transportation. Jackson [13] uses a similar example to introduce his problem frames. Truck WIM data is common in the transportation domain, where civil engineers use it for analyses such as road wear. A scale located in a traffic lane of a road weighs every axle that passes over it. It records the weight on the axle, the time of day, the lane the axle was in, and any error codes. Software components analyze this data to map axle data to vehicles, estimate the speed and length of the inferred vehicles, calculate a measure of load on an axle called ESAL (Equivalent Standard Axle Load), classify the vehicle type, eliminate passenger cars from the data, and (purportedly) filter out unreasonable values.

In our case study, a domain expert (the second author) interacted with the template mechanism to create a model of proper behavior for the WIM data feed from her informal expectations. These informal expectations can be summarized as: (1) vehicles in the same class should be similar and (2) vehicles should be physically feasible. Our method successfully turned these vague expectations into precise predicates. We used the resulting model for anomaly detection and compared it to existing documentation of the data feed. We show that the template mechanism is effective; we measure effectiveness both by the insights the user gains (the usefulness of the process) and the detection and misclassification rates (the usefulness of the resulting model).

2. The template mechanism of our approach

Our approach has three major stages: (1) setting up a model of proper behavior by eliciting precise user expectations; this stage relies on a novel template mechanism and is the focus of this paper, (2) using the precise expectations as a proxy for missing specifications to detect semantic anomalies in the data feed; previous work [22] discussed this stage, and (3) updating the precise expectations to account for evolving system behavior or user expectations; we defer this stage to future work.

These three stages may be viewed as a process governing the data and control flow among the mechanisms underlying our approach. These mechanisms are: (1) the *technique tool kit*—a collection of existing statistical and machine learning techniques that we support and adapt; Section 2.2 provides details, (2) the *template mechanism*—a mechanism that guides the human attention required in making expectations precise using templates that document the predicates a particular technique can output; Sections 2.1–2.2 provide details, and (3) the *anomaly detector*—a mechanism that

checks the predicates that are the precise user expectations and reports as anomalies data feed observations that falsify predicates. The anomaly detector utilizes the precise expectations as a model of proper behavior.

2.1. Process and premises

We characterize a predicate inference technique by the types of predicates it can produce. Templates capture the form of these predicates. For example, an inference technique may find a probable range for the values of a given attribute, e.g., the length attribute. The corresponding template would be $\# \leq \text{length} \leq \#$, where $\#$ is a numeric value.

The template mechanism operates as follows:

1. Select tool-kit techniques appropriate to the data and problem.
2. Run the selected techniques to infer predicates over subsets of the data.
3. Ask the user to classify each predicate as either “accept”, “update”, or “reject”.
4. Use the classification to instantiate templates.
5. Use the instantiated templates to filter the output of the tool kit techniques.
6. Give the filtered output to the anomaly detector and present to the user the resulting anomalies and their templates. Allow the user to change the classification.
7. Goto 2 or terminate when the user is happy with the classification.

An inferred predicate is a “complete instantiation” of a template. The template mechanism uses this complete instantiation for templates of “accept” predicates. Classifying a predicate as either “reject” or “update” may make the template instantiation partial by rendering the instantiation of all the numeric values in one or more dimensions void. See Section 2.2 for examples.

The template mechanism treats the predicate inference techniques as black boxes and uses the instantiated templates to filter the predicates a technique infers. It constructs and updates the model of proper behavior from instantiated templates of “accept” and “update” predicates. It will never present the user or the anomaly detector with predicates that match templates of previously rejected predicates. The template mechanism eliminates techniques that are not relevant for this user and data: it will not employ an inference technique if the user rejects all the predicates that are associated with this technique.

Premises of our template mechanism include (1) it is easier for a user to choose from a list of inferred predicates than to create this list, so having a machine synthesize the list is helpful and (2) it is easier for a user to understand expectations about data behavior when presented with examples. It is especially useful to examine examples of anomalous behavior, with the predicates that flagged them as anomalous.

2.2. Inference techniques and their templates

Our technique tool kit currently consists of five existing techniques that it supports and adapts: Rectmix (described below), Percentile (described below), K-means [19] (a clustering algorithm with hard membership), Association Rules [1] (a technique that produces probabilistic rules in an ‘if then’ form), and Daikon [10] (a program analysis tool that dynamically discovers likely invariants over program executions). We selected these techniques because they expose different aspects of the data and because their output is easy for a human to understand.

To select the most promising techniques for the problem, our method looks at the match between: (1) the data type and a technique (utilizing measurement scales [11]) and (2) the user expectations and the vocabulary of a technique. For the WIM data, this analysis found that the most promising techniques are Rectmix and Percentile: the predicates they output match the data types and describe data behavior relevant to the expert expectations. For this data feed, the other techniques either describe irrelevant behavior or produce predicates that are less precise or redundant with respect to the Rectmix and Percentile predicates. Therefore, we concentrate on the Rectmix and Percentile techniques and describe their templates. Details about the other techniques can be found in [20].

2.2.1 The Rectmix technique

Rectmix [18] is a clustering algorithm that supports soft membership (a point can probabilistically belong to multiple clusters). The clusters it finds are hyper-rectangles in N-space. Rectmix provides a measure of uncertainty called sigma (an estimate of the standard deviation) for each dimension. Anomalies are points that are not within a rectangle. Though clusters rarely have a hyper-rectangle shape in reality, Rectmix has the significant advantage of producing output that is easy to understand: a hyper-rectangle is simply a conjunction of ranges, one for each attribute (see Table 1). Rectmix has two parameters: the number of rectangles and the number of sigmas of uncertainty to allow.

Rectmix always outputs hyper-rectangles, so it has a single template: $\# \leq A_1 \leq \# \wedge \dots \wedge \# \leq A_n \leq \#$, where n is the number of attributes. The *dimensionality* of a template is the number of attributes in the template. Table 1 gives an example of user classification for predicates that Rectmix outputs for a subset of the WIM data. The corresponding templates have numeric values in one dimension—the axle attribute—because the user chose to void the other attribute values. For example, the template for the first predicate is $\# \leq \text{length} \leq \# \wedge \# \leq \text{ESAL} \leq \# \wedge 3 \leq \text{axles} \leq 3 \wedge \# \leq \text{weight} \leq \#$.

2.2.2 The Percentile technique

Percentile outputs a probable range for the values of each attribute. The x percentile of a distribution is a value in

Class	Length \wedge	ESAL \wedge	Axles \wedge	Weight
Update	20–42	0–.43	3–3	12–29
Update	23–44	0–1.2	2–3	26–47
Reject	13–100	0–.45	2–7	7–40
Update	23–29	0–6.7	2–4	27–71

Table 1. Example of Rectmix predicates classification

Class	Predicate	Template
Update	$40 \leq \text{speed} \leq 88$	$\# \leq \text{speed} \leq \#$
Update	$17 \leq \text{length} \leq 39$	$\# \leq \text{length} \leq \#$
Reject	$.06 \leq \text{ESAL} \leq .9$	$\# \leq \text{ESAL} \leq \#$
Update	$3 \leq \text{axles} \leq 3$	$\# \leq \text{axles} \leq \#$
Update	$12 \leq \text{weight} \leq 49$	$\# \leq \text{weight} \leq \#$

Table 2. Example of percentile predicates classification and instantiated templates

the distribution such that $x\%$ of the values in the distribution are equal or below it. Percentile calculates the range between the x and $100-x$ percentiles and allows $y\%$ uncertainty. Percentile only assumes that the distribution values are somewhat centered and is insensitive to extreme values.

Percentile has a single template: $\# \leq A \leq \#$. Table 2 gives an example of user classification and resulting instantiated templates for predicates that Percentile infers over a subset of the WIM data. Percentile ($x=25$, $y=25\%$) works well for speed, length, axles, and weight, but not for ESAL (ESAL seems to be exponentially distributed).

Rectmix and Percentile differ: Rectmix finds correlations among common attribute values whereas Percentile simply finds common values for a single attribute.

3. Case study hypothesis

The case study explores the hypothesis that the template mechanism is effective in eliciting precise user expectations and that the resulting precise expectations are a “good enough” engineering approximation to missing specifications, for the purpose of semantic anomaly detection.

The case study supports the hypothesis by showing that (1) The precise expectations are useful in detecting semantic anomalies in the WIM data and (2) The user gains insights about the WIM system through interaction with the template mechanism and through analysis of anomalies.

4. Data and methodology

In a WIM system multiple algorithms process raw sensor data, as introduced in Section 1. Unfortunately, processing and sensors are error prone. Errors may manifest as real vehicles that are not in their correct class (they are very different from other vehicles in their assigned class) or vehicles that are physically improbable. These are the kind of anomalies our expert cares about.

The data we use in our experiments is experimental data the Minnesota Department of Transportation collected by

its Mn/ROAD research facilities between January 1998 and December 2000. The data has over three million observations for ten vehicle types that characterize commercial vehicles. Vehicle types differ mainly by their number of axles and whether they consist of a single unit, a single trailer, or multi trailers. The number of observations the system collects varies by vehicle type.

We characterize the WIM data feed as a time-stamped sequence of observations. Each observation has attribute values for a single truck: date and time (accurate to the millisecond), vehicle type (one of ten classes), lane (one of two classes), speed (mph), error code (one of twenty five classes), length (feet), ESAL (dimensionless), number of axles, and weight (kips—kilo-pounds).

We first look for clusters and select attributes (details can be found in [20]). As a result, the template mechanism interacts with the user for each vehicle type (class) separately and gives the selected attributes to techniques in the tool kit.

For the purpose of validating our template mechanism, we selected three out of the ten vehicle types the data contained: the most common vehicle type (type 9, about two million observations) and two additional types (types 4 and 6, about one hundred thousand observations each).

A domain expert set up a model of proper behavior. We gave the model to the anomaly detector. To simulate the nature of on-line data, we divided the data into subsets of two thousand consecutive observations each.

5. Results

We briefly summarize the results of our case study. We present graphs and tables for one of the three vehicle types we examined (type 6). The results for the other two types (types 4 and 9) are rather similar.

The “update” predicates of Tables 1 and 2 are an example of precise user expectations for vehicle type 6.

The *detection rate* calculates how many attributes the model flags as anomalies out of the total number of attributes. It is an objective measure because the results of using the model for anomaly detection are binary: normal or anomalous.

Figure 1 shows the detection rate of the Percentile predicates. The analogous figure for Rectmix is similar.

The y-axis in a plot gives the total number of anomalies in one of the data subsets, according to the criterion the plot specifies, e.g., length anomalies. Notice that the y-axis scale differs among plots. The x-axis is the sequential subset index. The first column in Figure 1 summarizes the number of anomalies for each attribute. The plots in the second and third columns summarize the anomalies that are due to attribute values that are lower or higher, respectively, than the range bounds.

Table 3 summarizes the average detection rate over the subsets of each vehicle type. It gives the detection rate over

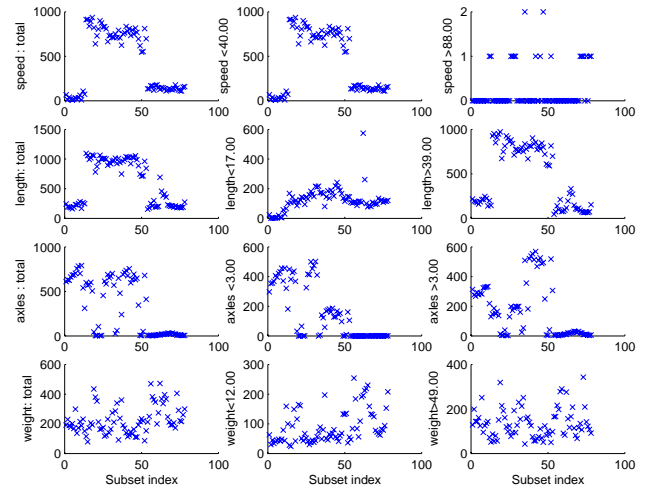


Figure 1. Counts of anomalies detected using Percentile predicates for vehicle type 6

Rectmix	Vehicle type	Average detection rate (%)					
		Total	Length	ESAL	Speed	Axles	Weight
	4	15.5	42.5	7.7		4.4	7.4
	6	10.9	37.7	0.4		0.6	4.8
	9	2.3	5.0	3.4		0.0	0.9
Percentile	4	8.4	8.1		0.8	10.2	14.6
	6	20.2	30.5		22.2	17.0	11.3
	9	0.8	1.0		0.3	0.0	1.9

Table 3. Average detection rate

all attributes and a break-down by attribute.

Small differences in the ranges for length and weight result in large differences in the detection rate, indicating that the values for these attributes are closely concentrated. The exact cut-off point between normal and anomalous is, therefore, not clear from the data.

The overall *misclassification rate* is defined as $\frac{FP+FN}{Nor+Ab} = \frac{Ab+FP-TP}{Nor+Ab}$ [23], where True Positives (TP) are correctly detected anomalous data, False Positives (FP) are normal data falsely detected as anomalous, False Negatives (FN) are undetected anomalous data, Normal (Nor=TN+FP) is data that is actually anomaly-free, and Abnormal (Ab=TP+FN) is data with actual anomalies.

Determining the above measures is subjective even though WIM documentation exists. This is because, on the one hand, the documentation is sometimes incomplete and imprecise, and on the other hand, it sometimes describes behavior that neither Rectmix nor Percentile can express.

To determine Ab, FP, and TP, our expert set constraints based on analyzing both the anomalies flagged by the anomaly detector and the differences between the inferred and documented models. Table 4 summarizes the resulting misclassification rate, averaged over the data subsets of

Vehicle type	Average misclassification rate (%)	
	Rectmix	Percentile
4	8.5	3
6	2.3	2.3
9	1	.8

Table 4. Average overall misclassification rate

each vehicle type. The rates are reasonable for a human to handle.

6. Analysis

The user gained insights by interacting with the template mechanism and by analyzing the resulting anomalies. This is an especially encouraging result because not only is our user a domain expert, but she also previously analyzed this data (though for a different purpose) [5]. In addition, the techniques inferred predicates that confirmed the expert knowledge about the system. This raised our confidence in the results and contributed to better understanding how the system works.

We first enumerate data behavior that surprised our expert. We then present her suggestions for explaining this behavior and enumerate the insights she gained by becoming aware of this behavior.

When looking at the anomalies detected by using her precise expectations as a model of proper behavior, the expert found the following data behavior surprising. This behavior is depicted in Figure 1. The data shows

- A large number of axle anomalies. In particular, the data shows a surprisingly large number of one axle vehicles. However, trucks should have at least two axles and the WIM system software should have detected such anomalies.
- A large number of slow vehicles.
- A large number of over-length vehicles. In particular, for type 6 vehicles, a large number of anomalies have the value of a system built-in length limit.
- A correlation between slow and over-length vehicles.
- A substantial decrease in the above anomalies starting with data subset number 54 (observed at Nov. 1999).
- An exception to all of the above for the most common truck type (type 9): the exceedingly large number of anomalies does not apply to it.

The expert suggested causes for this surprising behavior: The large number of anomalies may be due to (1) inaccurate physical sensing, (2) unintended interaction effects among the various software components. E.g., the component that should eliminate infeasible values—the filtering algorithm—may not properly clean the output of the component that should identify the vehicle type—the classification algorithm, and (3) boundary problems in the classification algorithm.

The decrease in the number of anomalies may be due to a software update in the classification or filtering algorithms or a re-calibration of the WIM scale. The similar behavior of multiple attributes and vehicle types suggests this change or update was system wide. The exception for the common vehicle type suggests that the system is tuned for this type.

The correlation between slow and over-length vehicles corroborates the expert knowledge.

The major insights our expert gained from the above analysis are as follows:

- The data behavior strongly suggests that there was a system-wide change in the WIM system starting November 1999.
- The system (both hardware and software) seems to be calibrated for the most common type of trucks. This, in turn, seems to adversely affect the accuracy of vehicle identification and classification of other types.
- The interaction of the various software components seems to occasionally have undesirable effects.

The data providers confirmed the expert insights and cause analysis, including the system wide change in Nov. 1999. They were unaware of the behavior that surprised our expert until recently. It turns out that the WIM scale has two different modes for weighing an axle. The various algorithms made inconsistent assumptions about the weigh mode. As a result, they occasionally assigned values to the wrong attribute. The next algorithms in the chain did not recognize the problem and made calculations based on the incorrect data. Type 9 vehicles are cleaner because one of the many software providers recognized a problem and made an undocumented correction for type 9. In addition, the system is physically calibrated for this type.

The above strengthens our belief in the usefulness of our method and demonstrates the benefits of automated elicitation support. To set up the model, the expert invested less than 10 hours. The anomaly detection was fully automated and quick (a few minutes). In comparison, it had taken the data providers several months to independently notice the same problems.

7. Related work

The main contribution of this paper is the template mechanism—a means of specifying user expectation and consequently checking these expectations to detect anomalies. Work most closely related includes approaches that either have a similar emphasis on users and their intent [16, 24, 15, 17] or perform various dynamic analysis based on observable behavior [10, 9, 2, 8, 14, 12]. However, that work often requires source code, binaries, or cooperation from the software providers and has a different domain.

We use existing unsupervised learning techniques. Co-training [4] tries to reduce the effort that labeling data for supervised learning requires. Active learning [7] tries to

select good training data for a technique. We ask the user to classify the output of a technique, rather than its input.

Many people have been analyzing WIM data. However, most are concerned with transportation issues, not data quality. [6, 5] did domain specific quality analysis.

8. Conclusions

We introduced a promising means for eliciting user expectations about data behavior: the template mechanism. Our case study provides empirical evidence in support of the effectiveness of the template mechanism: (1) The model was useful for anomaly detection. It enabled detecting actual anomalies that the expert cared about: classification problems and unlikely vehicles. In addition, the misclassification rate was reasonable for a human to handle. (2) The expert gained insights about the WIM system. The data providers confirmed the expert insights.

Moreover, the case study results corroborate the benefits of interacting with the template mechanism to make expectations precise and of analyzing the resulting anomalies. Our method: (1) detected hardware and software problems from observed data only. It detected, for example, problems that were caused by mis-calibration, software modifications, or state changes, (2) promptly detected these problems, and (3) increased the understanding of existing documentation. For example, the exact cut-off point between normal and anomalous was not clear from the data though it was clear (for upper bounds) from the documentation, suggesting the documentation bounds may be too strict.

9. Acknowledgments

We thank the Auton Lab [3] for making their dataset processing and analysis software (SPRAT) available to us, the Minnesota Department of Transportation for their WIM data, and Dan Pelleg for allowing us to use his Rectmix code and for his comments. This research is supported by NSF under Grant ITR-0086003, by the Sloan Software Industry Center at Carnegie Mellon University, by the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298, and by the General Motors Collaborative Research Laboratory at Carnegie Mellon. This material is based in part upon work supported by the National Science Foundation under Grant Number 9987871, and by the EUSES Consortium via the National Science Foundation (ITR-0325273).

References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD* 93, 1993.
- [2] G. Ammons, R. Bodik, and J. Larus. Mining specifications. In *POPL*, 2002.
- [3] Auton Lab. URL: <http://www.autonlab.org>. Accessed April 2003.
- [4] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Workshop on Computational Learning Theory*, 1998.
- [5] R. Buchheit. *Vacuum: Automated Procedures for Assessing and Cleansing Civil Infrastructure Data*. PhD thesis, Carnegie Mellon University, Civil Engineering Dept., 2002.
- [6] R. Buchheit, J. Garrett Jr., S. McNeil, and M. Chalkline. Automated procedures for improving the accuracy of sensor-based monitoring data. In *AATT*, 2002.
- [7] D. Cohn, Z. Ghahramani, and M. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.
- [8] W. Dickinson, D. Leon, and A. Podgurski. Finding failures by cluster analysis of execution profiles. In *ICSE*, 2001.
- [9] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as deviant behavior: A general approach to inferring errors in systems code. In *18th ACM Symposium on Operating Systems Principles*, 2001.
- [10] M. Ernst, J. Cockrell, W. Griswold, and D. Notkin. Dynamically discovering likely program invariants to support program evolution. In *TSE*, 2000.
- [11] N. E. Fenton and S. L. Pfleeger. *Software Metrics*, chapter 2. PWS Publishing Company, 2nd edition, 1997.
- [12] S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. In *Evolutionary Computation Journal*, 2000.
- [13] M. Jackson. *Problem Frames: Analysing and Structuring Software Development Problems*, chapter 4.3.3, 5.4. Addison-Wesley, 2001.
- [14] T. Lane and C. E. Brodley. Approaches to online learning and concept drift for user identification in computer security. In *KDD*, 1998.
- [15] P. Langley. The computational support of scientific discovery. *International Journal of Human-Computer Studies*, 53, 2000.
- [16] S. McCamant and M. Ernst. Predicting problems caused by component upgrades. In *ESEC/FSE*, 2003.
- [17] R. C. Miller and B. A. Myers. Outlier finding: Focusing user attention on possible errors. In *UIST*, 2001.
- [18] D. Pelleg and A. Moore. Mixtures of rectangles: Interpretable soft clustering. In *ICML*, 2001.
- [19] P. H. R. Duda and D. Stork. *Pattern Classification*,. John Wiley and Sons, 2nd edition, 2000.
- [20] O. Raz, R. Buchheit, M. Shaw, P. Koopman, and C. Faloutsos. Eliciting user expectations for data behavior via invariant templates. Technical report, CMU-CS-03-105, 2003.
- [21] O. Raz, R. Buchheit, M. Shaw, P. Koopman, and C. Faloutsos. Detecting semantic anomalies in truck weigh-in-motion traffic data using data mining. *Journal of Computing in Civil Engineering (JCCE)*, 2004. Accepted.
- [22] O. Raz, P. Koopman, and M. Shaw. Semantic anomaly detection in online data sources. In *ICSE*, 2002.
- [23] P. Runeson, M. Ohlsson, and C. Wohlin. A classification scheme for studies on fault-prone components. In *Product focused software process improvement*, 2001.
- [24] J. Sousa and D. Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. In *IEEE/IFIP Conference on Software Architecture*, 2002.

Automated Support for Knowledge Engineering for

A Natural Gas Pipeline Domain

Christine W. Chan
Software Systems Engineering
Faculty of Engineering
University of Regina
Regina, Sask S4S 0A2
Canada
Email: Christine.Chan@uregina.ca

Abstract This paper presents application of the Knowledge Modeling System that supports users in documenting knowledge elements acquired for development of knowledge-based systems. The system was developed based on a knowledge modeling technique called the Inferential Modeling Technique. It facilitates building an application ontology of a domain by explicitly storing and structuring both domain and task knowledge elements of any industrial problem domain. The system also can support knowledge sharing by converting the ontology into XML. Application of the system for constructing an application ontology in the natural gas pipeline domain is also presented.

Keywords: knowledge modeling, ontology construction, natural gas pipeline operations, expert systems

1. Introduction

This paper presents the Knowledge Modeling System that supports users in building an application ontology from knowledge acquired for developing a knowledge based system. Knowledge acquisition is the process that extracts the required knowledge from available sources, such as experts, textbooks and databases, for incorporation into a knowledge-based system. This is an acknowledged bottleneck in the development of knowledge based systems. But it is also a crucial process in the development of knowledge-based expert systems because quality of the elicited and represented knowledge determines effectiveness and efficiency of the knowledge based system eventually constructed. The problem of modeling knowledge is especially difficult for knowledge engineers faced with the task of creating a knowledge model for a domain with which they are unfamiliar.

This paper presents application of a tool for knowledge modeling called the Knowledge Modeling

System (KMS) to a natural gas pipeline domain. The tool was developed based on the Inferential Modeling Technique [1,2]. The objective of developing the knowledge modeling tool is to provide automated support for representing knowledge obtained during knowledge acquisition as a step towards construction of an application ontology. An ontology is an explicit specification of a conceptualization that provides a comprehensive foundation specification of knowledge in a domain. The Knowledge Modeling System implemented based on the Inferential Modeling Technique supports acquiring and storing knowledge items on a domain. In the current version, the repository of knowledge items resides in a relational database, which can be translated into the sharable representation of XML. The knowledge model expressed in XML can function as a sharable and reusable ontology of an application domain.

2. Background

Different definitions of ontology have been proposed. An ontology can be defined as a description of the most useful, or at least most well-trodden, organization of knowledge in a given domain. The organization involves explicit specification of the objects, and relationships that make up some world [6]. An ontology can also be considered as an explicit specification of a conceptualization. It provides a comprehensive foundation specification of knowledge in a domain. In the simplest case, an ontology can be represented as a hierarchy of concepts related by subsumption relations. In more complex cases, a variety of axioms can be added to express relationships and constraints among domain concepts. Ontological analysis as a knowledge modelling technique was used in various research efforts. For example, the CommonKADS methodology suggested knowledge categorization in the model of expertise to

consist of the two major types of domain theory and control knowledge, the latter includes inference, task, and strategic knowledge (see e.g. Flores-Mendez et al.[4]). Within this field of practical knowledge level modeling, the Inferential Modeling Technique (IMT) supports developing knowledge level models for diverse task and domains. Similar to the CommonKADS methodology, the IMT emphasizes both domain and task specific elements [1,2]. This technique suggests modelling the domain objects and relations first before deciding what tasks are involved and what problem-solving methods to adopt. Assigning first priority to modeling domain objects and relations does not connote these are more important knowledge elements than the task knowledge. Both domain and task characteristics are equally significant and intricately related.

3. The Inferential Modeling Technique

The Inferential Model and Inferential Modeling Technique have been presented in [1,2]. The high level content theory implicit in the Inferential Modeling Technique provided the theoretical basis for developing the Knowledge Modeling System, and implicit assumptions of the Inferential Modeling Technique also guided formulation of the Knowledge Modeling System. The Inferential Modeling Technique is presented as follows.

The template of knowledge types specified by the Inferential Model serves to guide the elucidation of the search space for a given problem domain. The model can be operationalised as a procedure which facilitates the development of the "specific categories" for a given domain by presenting the knowledge engineer (KE) with a template of knowledge types and a sequence of steps whereby the elicited units can be classified. The Inferential Modeling Technique consists of the following steps:

1. specify the physical objects in the domain,
2. specify the properties of objects identified in Step 1,
3. specify the values of the properties identified in Step 2, or,
4. define the properties as functions or equations,
5. specify the relations associated with objects and properties identified in Steps 1 and 2 as functions or equations,
6. specify the partial order of the relations identified in Step 5 in terms of strength factors and criteria associated with the relations,
7. specify the inference relations derived from objects and properties identified in Steps 1 and 2,
8. specify the partial order of the inference relations identified in Step 7 in terms of strength factors and criteria associated with the relations,

9. specify the tasks in the problem,
10. decompose the tasks identified in Step 9 into inference structures or subtasks (which invoke units identified in Steps 1, 2, 5, and 7),
11. specify the partial order of the inference and subtask structures identified in Step 10 in terms of strength factors and criteria,
12. specify strategic knowledge in the domain,
13. specify how strategic knowledge identified in Step 12 is related to task and inference structures specified in Steps 9 and 10,
14. return to Step 1 until the specification of knowledge types is satisfactory to both the expert and KE.

This procedure supports an iterative-refinement of the knowledge acquired for a problem domain and provides top-down guidance on the knowledge types that are required for problem solving. The termination of this procedure occurs when both the knowledge engineer and expert are reasonably satisfied that the knowledge model that emerges represents the problem solving expertise.

4. Design of the Knowledge Modeling System

The IMT provided the theoretical basis for developing the Knowledge Modeling System, which supports construction of an application ontology. The Knowledge Modeling System (KMS) has been designed to support acquiring and storing both static and dynamic knowledge elements of a domain. The two modules correspond to the orthogonal axis of static and dynamic knowledge of a problem domain as specified in the IMT. The two components are complementary and together can adequately represent most types of knowledge implicit in an industrial application domain. The system accepts user input through the user interface to either the class or task component. All the knowledge obtained from the user are stored in relational database tables implemented as .mdb files in MS Access (trademark of Microsoft). The databases can be converted into XML format for knowledge sharing and dissemination on the web. The two main components of KMS are described as follows.

4.1. Class component

The class component corresponds to the domain and inference levels in the IMT and consists of the domain and inference classes, attributes, and values. This component elicits from the user static knowledge on an application domain such as classes of objects, the attributes and values associated with each class, and relationships between the classes. The classes specified can be referring to either concrete or conceptual entities in the real world. The KMS also supports specification of binary relationships between classes. In the current version of the system, only the inheritance or isa relationship between parent and child classes is

supported. Hence, the attributes specified for the parent or super class can be inherited by the children or subclasses. In addition, a class can also have its own specific attributes. Based on the inheritance relationships, the KMS can automatically configure a classification hierarchy of all the classes and subclasses in the domain. In the current version of the KMS, only binary relationships between two classes are supported. If a class is involved in a ternary or higher order relationship, it is specified as a “constraint” in text.

4.2. Task Component

The task component of the system elicits knowledge about the dynamic aspect of an application domain. Based on the IMT, a task is regarded as an organized structure or sequence of activities that is performed to accomplish some objective. In KMS, objectives and tasks are independent and managed separately. A task is linked to an objective provided the latter needs the former to complete itself. The detailed steps involved in a task structure are described as behaviour. Similar to the hierarchical structure relating classes and subclasses, tasks are organized into a hierarchical structure so that a task can be divided into subtasks. Again similar to the notion of attributes for classes, properties can be defined in KMS to describe tasks. Some sample characteristics associated to a task include its preconditions, dependencies, objects involved in a task and its documentation. A task can also be associated to more than one objective. The percentage of completion states the extent to which the task is finished. The behavior of the task specifies the steps involved in the task structure needed to complete the task, this can be defined in pseudo code.

4.3. Interaction between class and task components

According to the IMT, the dynamic knowledge of a problem domain is intricately intertwined with the static knowledge. That is, the tasks and subtasks manipulate classes of objects in order to accomplish an objective. In KMS, the interaction between tasks and classes is implemented as the task component invoking particular class objects defined in the class component of the tool. In applying the IMT, the knowledge engineer first defines the static knowledge elements of a domain before the dynamic knowledge elements. Similarly, users of KMS need to first define the classes of objects, their attributes and values in the class component. Then, the task component of the system can invoke specific objects that belong to classes already defined in the system and instantiate the tasks with objects.

5. Application Problem Domain

In natural gas pipeline operations, a dispatcher is responsible for making two vital decisions: (1) increase and decrease compression, and (2) select individual compressor units to turn on/off. These decisions have a significant impact on effectiveness of the natural gas pipeline operation. When the demand for natural gas customers increases, the dispatcher adds compression to the pipeline system by turning on one or more compressors; and when customer demand for natural gas decreases, the dispatcher turns off one or more compressors to reduce compression in the pipeline system.

To better focus the development efforts, a small section of the natural gas pipeline in Saskatchewan Canada called the St. Louis East compressor station was modeled. A schematic of the St. Louis East system is shown in figure 1.

The system consists of two compressor stations, Melfort and St. Louis. These compressor stations are used to supply natural gas to two customer locations, Nipawin and Hudson Bay. In St. Louis, there are three compressor units. Two of these units are electrical compressor units and the other is a gas compressor unit. In Melfort, there are two gas compressors. An electrical compressor unit provides 250 horsepower and a gas unit provides 600 horsepower. The demand for natural gas from the customers fluctuates depending on the season. In the winter, the demand for natural gas is usually higher than in the summer. In addition, the demand for natural gas also changes depending on the time of day.

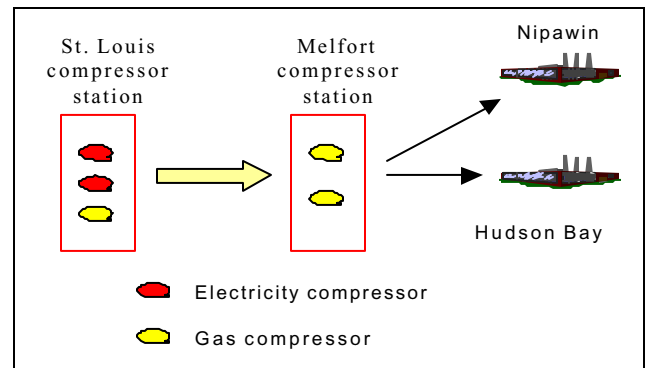


Figure 1 Schematic of the St. Louis East system

Knowledge modeling was conducted for developing an expert decision support system called Gas Pipeline Operations Advisor (GPOA). It can aid the dispatcher in optimizing natural gas pipeline operations in order to satisfy customer demand with minimal operating costs. The purpose of GPOA is to inform the dispatcher whether compression should be added or reduced in a pipeline system and the horsepower requirement needed to satisfy customer demand, based on the total inline

flows and the current system conditions. In the process of knowledge acquisition, the expert dispatchers suggested a primary consideration in pipeline operations was linepack level, which is a key variable used to measure the value of the comfort zone. Linepack is defined as the volume of natural gas that exists between the compressor discharge pressure and the customer end-point delivery pressure.

Some key concepts identified in this domain consisted of the following. First, the four major conditional variables in pipeline operations include rate of change of pressure at the end point, current linepack level, change of pressure at the end point, total flow in the pipeline, and the decision variable of the state of the linepack, which measures the value of the comfort zone. Secondly, in the St. Louis East subsystem, there are two types of compressors and a total of five compressors. The dispatcher operates them to control pressure in the pipeline system. Thirdly, the demand for natural gas is higher in the winter than summer. The time of the day also affects the demand for natural gas.

After acquiring the knowledge, the IMT was applied for knowledge analysis. The IMT provides a template of the possible knowledge types in a domain, and supports the knowledge engineer in identifying the knowledge types in the gas pipeline domain. According to the IMT, some sample knowledge elements in this domain include the following:

- A class of objects: A class of concrete or abstract objects, e.g. a pipeline is a concrete class, which is a medium between a compressor station and customers,
- an attribute: an attribute describes a class, e.g. temperature is a property that describes the state of the pipeline,
- value: a value for an attribute can be numeric or symbolic; in the natural gas pipeline network operations domain, a value can be a numeric or logical (Boolean) value. An example of a numeric value is capacity of a gas compressor at St. Louis station is 600 BHP. An example of a logical value is status of the gas compressor at St. Louis, which is either on or off.
- relation: a relation between two or more classes of objects, e.g. the inheritance relationship is a relationship between an electrical compressor and its parent, the compressor, and it is expressed as “an electrical-compressor isa compressor”
- task: a task is a set of activities that accomplish an objective, e.g. the task of compressor selection involves selecting a compressor in order to put additional pressure into the pipeline.

These knowledge elements were explicitly documented in KMS, and were configured into an application ontology.

6. Knowledge Representation Using KMS

The knowledge elements clarified using IMT provided the basis for an ontology of the domain. Figure 2 shows a sample input screen of KMS that allows a knowledge engineer to enter information on classes, sub-classes, attributes and values of objects belonging to a domain. The top left panel of the screen shown in figure 2 shows the classes and subclasses. For example, the class of pipeline has the subclass of gas pipeline, and the class of supplier station has the subclass of gas supplier station, etc. All the classes are listed on the lower left panel of the screen. The highlighted class is gas-pipeline, and its class-specific and inherited attributes are listed on the lower left panel of the screen. The inherited attributes are prefixed with “#”. The attribute of “change of pressure at end point (COP)” is highlighted, and its possible values are listed in the top right panel of the screen.

Task knowledge can also be represented in KMS. To determine the BHP requirement, the linear equation used was: $BHP = 277411 \times (\text{St. Louis Flow} + \text{Melfort Flow}) - 1132$. This equation was provided by the domain experts.

For example, if the load is 900×103/day, the BHP requirement can be calculated to be 1400. In addition, the dispatcher can also consult the following prioritized list of compressors to be turned on at different ranges of BHP requirement (where G1 is gas compressor numbered 1 and E1 is electrical compressor numbered 1 etc.):

1. Free flow (no compression)
2. ($0 < BHP = 800$) St. Louis G1
3. ($800 < BHP = 1200$) St. Louis G1 and Melfort G2
4. ($1200 < BHP = 1600$) St. Louis G1, Melfort G2, and St. Louis E1
5. ($1600 < BHP = 2000$) St. Louis G1 and E1, Melfort G2 and G3

The prioritized list of compressors constitutes important information for the dispatcher, and it was documented in the task component of KMS. The top left panel of the screen in figure 3 shows the task objectives in the problem domain of monitoring and control of gas pipelines. All the tasks involved in the highlighted objective of “determine horsepower requirement” are listed in the middle panel of the screen. On the top right panel, a decomposed list of the tasks and subtasks in the domain are shown. For example, the second task in the top right panel states “compressor selection”. The subtask under this is the prioritized list of compressors. By clicking on “follow the order from 1 to 5 based on BHP requirement”, the prioritized list for operating the compressors at the two stations is displayed in the bottom right panel of the screen.

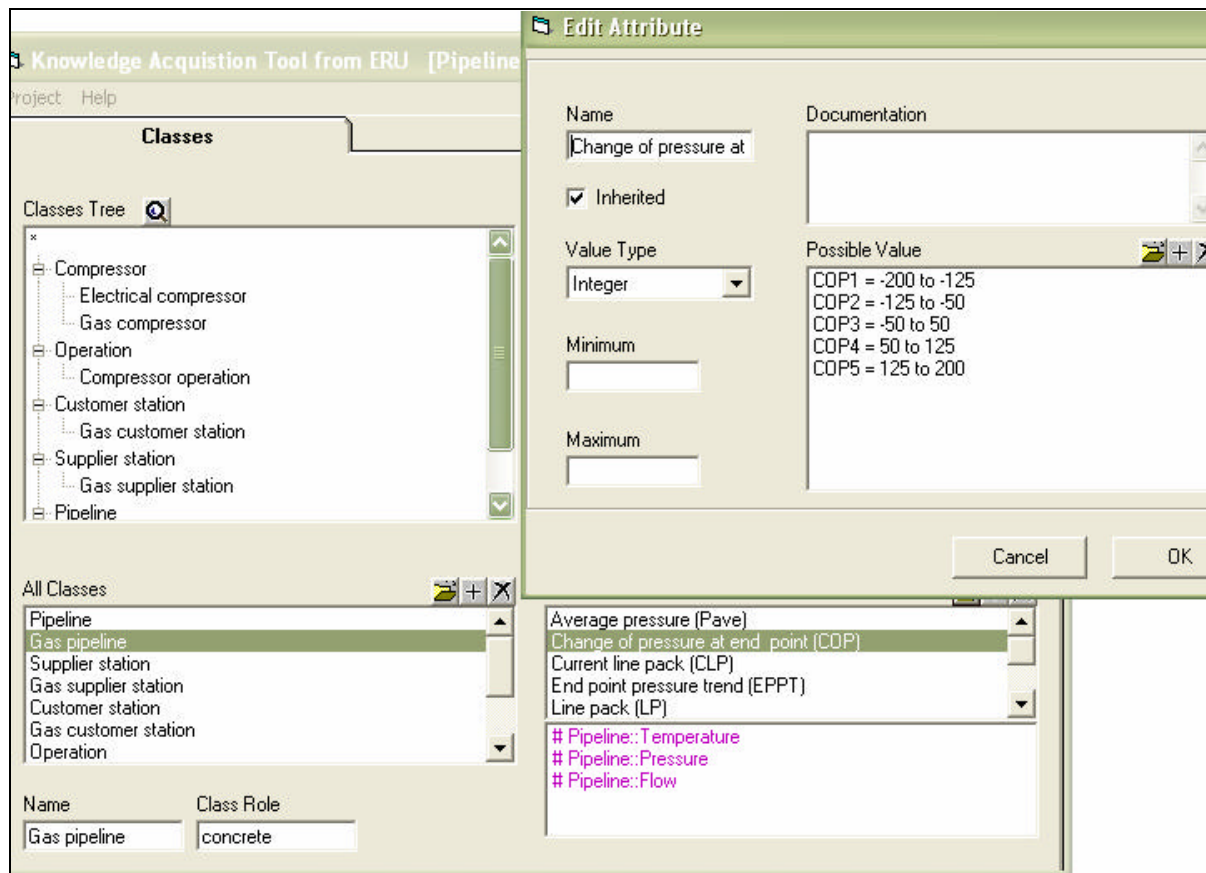


Figure 2 Representation of classes, attributes and values in KMS

7. Conclusion and Future Work

This paper has presented application of KMS for construction of an application ontology from acquired expertise. While the system can convert the specified knowledge into XML format, this function is not fully developed and needs further refinement. The KMS can be compared to other ontology engineering tools along the three dimensions of (1) development methodology, (2) use of ontology, and (3) software usability issue as suggested in [7]. Similar to other ontology engineering tools like OntoEdit that is based on On-To-Knowledge [8], and WebODE that is based on Methontology [3], the KMS is solidly grounded in the Inferential Modeling Technique which provided the theoretical basis for developing the tool. In terms of sharing of ontology, KMS is similar to OntoEdit, Hozo and WebODE in that they all support conversion to some sharable formalisms such as XML, DAML+OIL, or RDF. Although this feature needs refinement, it is functional in the current version. In terms of usability of the interface, KMS is similar to the other ontology

engineering tools in that it has sophisticated interface capabilities.

The key role that an ontology assumes in knowledge modeling and knowledge based system development has been widely discussed (see for example [5]). Mark et al. [6] suggested that an ontology can serve as software specification in knowledge-based system development. Like software architecture, an ontology provides guidance to the development process. The former provides guidance to the development process by specifying the interdependencies that deal with stages or aspects of a problem-solving process. By contrast to a software architecture, however, an ontology involves not only the stages of a process, but also the taxonomy of knowledge types. The two aspects are referred to as task-specific and domain-specific architectures [6]. There is much room for improvement in the current version of KMS. For example, more features can be incorporated into the system such as support for strength factors on relationships, facilities for modeling uncertainty in both static and dynamic knowledge, representing non-binary relationships among classes, and manipulation of attributes by the task component. These will be left for future research.

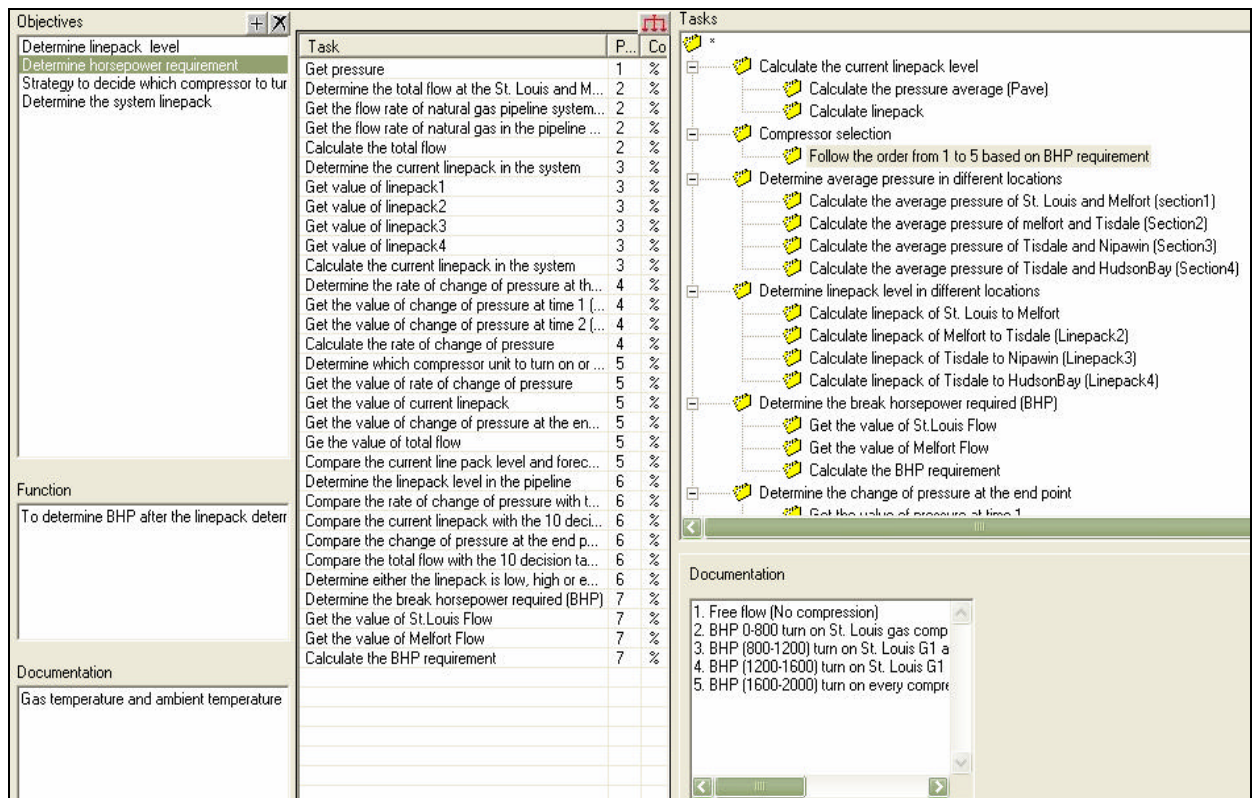


Figure 3 Representation of task knowledge in KMS

Acknowledgements

The author would like to thank W. Jin and V. Uraikul for their contributions to this work, and also would like to acknowledge the generous support of a Research Grant and a Strategic Grant from the Natural Sciences and Engineering Research Council of Canada.

References

- [1] C.W. Chan, "Development and Application of A Knowledge Modeling Technique", Journal of Experimental and Theoretical Artificial Intelligence, Vol. 7, No. 2, 1995, pp. 217-236.
- [2] C.W. Chan, "A Knowledge Modelling Technique and Industrial Applications", invited book chapter, Chapter 34 in Volume 4 of C. Leondes, Ed., Knowledge-Based Systems Techniques and Applications, 4 volumes Academic Press, USA, 2000.
- [3] O. Corcho, M. Fernandez-Lopez, A. Gomez-Perez and O. Vicente, "WebODE: An integrated workbench for ontology representation, reasoning, and exchange", Proc. of EKAW 2002, Springer LNAI 2473, 2002, pp. 138-153.
- [4] R.A. Flores-Mendez, P. van Leeuwen, and D. Lukose, "Modeling expertise using KADS and MODEL-ECS", in Proceedings of Banff Knowledge Acquisition Workshop '98, Banff Canada, October 1998.
- [5] N. Guarino and R. Poli, eds., Formal Ontology for Conceptual Analysis and Knowledge Representation, Dordrecht/Norwell, MA: Kluwer Academic, 1994.
- [6] W. Mark, J. Dukes-Schlossberg, and R. Kerber, "Ontological commitment and domain specific architectures: experience with Comet and Cosmos", In Towards Very Large Knowledge Bases, Amsterdam, Tokyo:IOS Press, Ohmsha, 1995.
- [7] R. Mizoguchi, "Ontology Engineering Environments", In Handbook on Ontologies, S. Staab and R. Studer, eds., 2003, pp. 275-295.
- [8] Y. Sure, S. Staab, M. Erdmann, J. Angele, R. Studer and D. Wenke, "OntoEdit: Collaborative ontology development for the semantic web", Proc. of ISWC 2002, 2002, pp. 221-235.

Automatic bug triage using text categorization

Davor Čubranić

Department of Computer Science
University of British Columbia
201–2366 Main Mall
Vancouver, BC, V6T 1Z4
cubranic@cs.ubc.ca

Gail C. Murphy

Department of Computer Science
University of British Columbia
201–2366 Main Mall
Vancouver, BC, V6T 1Z4
murphy@cs.ubc.ca

Abstract

Bug triage, deciding what to do with an incoming bug report, is taking up increasing amount of developer resources in large open-source projects. In this paper, we propose to apply machine learning techniques to assist in bug triage by using text categorization to predict the developer that should work on the bug based on the bug's description. We demonstrate our approach on a collection of 15,859 bug reports from a large open-source project. Our evaluation shows that our prototype, using supervised Bayesian learning, can correctly predict 30% of the report assignments to developers.

1 Introduction

Large software development projects require a bug tracking system to manage bug reports and developers who work on fixing them. A ubiquitous example of such a system is Bugzilla,¹ an open-source system first introduced in the development of the Mozilla web browser, but now used in numerous other projects.

Bug tracking systems are particularly important in open-source software development, where the team members can be dispersed around the world. In such widely-distributed projects, the developers and other project contributors may rarely, if ever, see each other. Consequently, the bug tracking system is used not only to keep track of problem reports and feature requests, but also to coordinate work among the developers.²

Most bug-tracking systems allow posting of additional comments in bug reports. With communication channels

between open source team members limited by their geographical and time separation, this feature has evolved to fill a niche for focused, issue-specific discussion. The comments on the bug report serve as forum for discussion of implementation details or feature design alternatives. Developers who can help in design deliberations because of their expertise and insight, and stakeholders whose code will be impacted by the proposed modifications, or who will have to implement and integrate them, are quickly brought into the discussion by “CC-ing” them on the bug report.³ Other members of the project with interest in the issue, often users who urgently need the feature or the bug fix, also join in. More contentious issues—usually requests for new features—can take months to resolve and can involve over a hundred comments from dozens of people.

In many ways, the bug tracking system is the public face that an open source development team presents to its user community. Therefore, it is important that new bug reports be dealt with as quickly as possible. Few things will turn the users away—and kill the project's community—faster than the perception that the developers are not responsive and ignore the users' bug reports and feature requests.

However, successful large open source projects are faced with the challenge of managing the incoming deluge of new reports.⁴ Effectively deciding what to do with a new report—*bug triage* in Mozilla parlance—can be a problem: it takes time to figure out whether the report is a real bug or a feature worth considering, to check that it is not a duplicate of an existing report, and to decide which developer should work on it. Past a certain rate of new bug reports, the time commitment for triage becomes too much of a burden for an experienced developer, whose attention is more valuable elsewhere. Projects such as Mozilla and Eclipse⁵

¹<http://www.mozilla.org/projects/bugzilla>.

²The bug tracking system therefore serves to track more than just bugs, and it may be more appropriate to call it “issue tracking system”. We use the terms “bug tracking system” and “bug report” for historical reasons, but in their wider, all-inclusive, sense.

³All developers on the CC list for a given bug report are automatically emailed notifications of changes to the report's status and new comments.

⁴The Mozilla project has received an average of 168 new bug reports per day in the week of 9 February 2004, for example.

⁵An extensible integrated development environment developed by IBM

have therefore been forced to introduce team members who are dedicated to bug triage [2]. This solution is not ideal, however, because it requires an additional step before the developer can start working on a bug. It also introduces potential errors, and more delays, if the triager makes a wrong decision to which developer to assign the report.

In this paper we present our investigation of using machine learning, and in particular text categorization, to “cut out the triageman” and automatically assign bugs to developers based on the description of the bug as entered by the bug’s submitter. The method would require no changes to the way bugs are currently submitted to Bugzilla, or to the way developers handle them once the bugs are assigned. The benefit to software development teams would be to free up developer resources currently devoted to bug triage, while assigning each bug report to the developer with appropriate expertise to deal with the bug.

We begin this paper with a brief overview of related work, followed by an introduction to the classification framework used and the theory behind it. We then present an experiment in which we applied these techniques to a selection of bug reports from the Eclipse project and tested their accuracy in assigning reports to developers. We conclude the paper with a discussion of results and possible avenues for future work.

2 Related work

We are aware of no other work on computer-assisted bug report triage, although there are some key insights on the interrelationship between bug reports, source code, and the developers that we share with the following two projects:

Fischer et al. mapped program features to the source code where they were implemented, and then tracked the code changes against problem reports involving those features [4]. They then visualize the established relationships to search for feature overlap and dependencies. Such visualization of the evolution of features across time can then be used to find locations in the code where there may be erosion in the software architecture of the system, indicating future problem spots for software maintenance.

Bowman and Holt have analyzed which developers worked on each file in a software system to determine its *ownership architecture* [1]. The ownership architecture complements other types of architectural documentation. It identifies experts for system components, and can be used to infer the project’s internal organization into sub-teams. The ownership architecture can also show non-functional dependencies: in their example device drivers for a given architecture could be easily seen, even if they otherwise shared no code and resided in separate portions of the filesystem

hierarchy, because they were “owned” by the same small group of developers.

Although our purpose is different, our approach bridges Bowman and Holt’s idea that there is a correspondence between a system’s components and individual developers with that of Fischer et al. on the link between bug reports and program features. We also note that all three projects are for support of *managing* software development, even if they mine the source code for the relevant information.

Machine learning and data mining techniques have already been applied to source code and program failure reports, although so far only to support the code-writing/debugging component of the software development effort. For instance, Zimmermann et al. mined source version histories to determine association rules which can then be used to predict files (or smaller program elements, such as functions and variables) that usually change together [11]. Such predictions can help prevent errors due to incomplete changes or show program couplings that wouldn’t be visible to methods such as program dependency analysis.

Also, Podgurski et al. use machine learning to cluster software failure reports to automatically determine which ones are likely to be manifestations of the same error [9]. The failure reports in this case are automatically generated, unlike the bug reports we deal with, and consist of stack traces at the moment of program crashes.

3 Classification framework

We treat the problem of assigning developers to bug reports as an instance of text classification, or “the problem of assigning a text document into one or more topic categories or classes” [6]. More specifically, it is a *multi-class, single-label classification* problem: each developer corresponds to a single class, and each document (that is, a bug report) is assigned to only one class (that is, a developer working on the project). Furthermore, it is a *supervised learning* problem, since we can view the correspondences of developers with the bugs that they fixed in the past as the training data.

A variety of techniques for supervised learning have been applied to text classification in recent years, for example: regression models, k-nearest neighbour, Bayes belief networks, decision trees, support vector machines, and rule-learning algorithms. (See Yang [10] for an overview of these approaches and a comparative evaluation of their performance.) In this paper, we report on the use of Bayesian learning approach for this project, because it is conceptually elegant, is easily adapted to multi-class classification, and performs well. The algorithm used, introduced by Kalt [5] and further developed by Nigam et al. [8] is presented in the following section, followed by the explanation of how we applied it in the bug triage domain.

3.1 Naive Bayes classifier for multinomial word-document model

The following are the framework’s assumptions: the data set is represented as a collection of documents, $\mathcal{D} = \{d_1, \dots, d_{|\mathcal{D}|}\}$, and each document has a class label $c \in \mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$. Documents in \mathcal{D} are generated by a mixture model and there is a one-to-one correspondence between mixture components and classes in \mathcal{C} . Mixture components, in turn, are parametrized on θ . Therefore,

$$P(d_i|\theta) = \sum_{j=1}^{|\mathcal{C}|} P(d_i|c_j, \theta)P(c_j|\theta) \quad (1)$$

Furthermore, the documents are represented as “bags of words”: each document d_i consists of words w_t drawn from vocabulary $\mathcal{V} = \{w_1, \dots, w_{|\mathcal{V}|}\}$. The *naive Bayes* assumption is that the words are independently and identically distributed (i.i.d.): the probability of each word is independent of its context and position in the document. Thus, the documents are drawn from a multinomial distribution:

$$P(d_i|c_j, \theta) = \prod_{t=1}^{|\mathcal{V}|} \delta_{tj}^{N_{ti}} \quad (2)$$

where $\delta_{tj} = P(w_t|c_j, \theta)$ and N_{ti} is the number of times word w_t occurs in the document d_i . While the naive Bayes assumption is clearly false in many real-world situations, classifiers based on it perform surprisingly well, and it turns out that it is not a mathematically unreasonable assumption to make in classification tasks [3].

Using the Bayes rule, a previously unseen document d_i can then be assigned label c_j which maximizes:

$$P(c_j|d_i, \theta) = \frac{P(c_j|\theta)P(d_i|c_j, \theta)}{P(d_i|\theta)} \quad (3)$$

$$\propto P(c_j|\theta) \prod_{t=1}^{|\mathcal{V}|} \delta_{tj}^{N_{ti}} \quad (4)$$

The priors are estimated from the training data:

$$P(c_j|\theta) = \frac{\sum_{i=1}^{|\mathcal{D}|} P(c_j|d_i)}{|\mathcal{D}|} \quad (5)$$

where $P(c_j|d_i) = \{0, 1\}$ as given by the training set labels (that is, $P(c_j|\theta)$ equals the number of times c_j occurred in the test set divided by the size of the test set); and

$$P(w_t|c_j, \theta) = \frac{\sum_{i=1}^{|\mathcal{D}|} N_{ti}P(c_j|d_i)}{\sum_{m=1}^{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{D}|} N_{ti}P(c_j|d_i)} \quad (6)$$

(That is, $P(w_t|c_j, \theta)$ equals the number of times word w_t occurs in class c_j divided by the total number of all word

occurrences in that class.) In practice, a Laplace prior is often used to avoid zero probabilities of words occurring infrequently in \mathcal{V} and was used here as well:

$$P(w_t|c_j, \theta) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} N_{ti}P(c_j|d_i)}{|\mathcal{V}| + \sum_{m=1}^{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{D}|} N_{ti}P(c_j|d_i)} \quad (7)$$

3.2 Bug triage as a Naive Bayes classifier

Our dataset \mathcal{D} is a collection of bug reports $\{d_1, \dots, d_{|\mathcal{D}|}\}$ entered into the bug tracking database. When a new bug report is submitted, it is given a one-line summary and a longer description. The bug report d_i thus consists of a set of words w_t that appear in its summary and description. The order of words does not matter, but we do keep track of multiple occurrences of a word in a single bug report, N_{ti} .

The developers working on the project form our set of classes $\mathcal{C} = \{c_1, \dots, c_{|\mathcal{C}|}\}$. Although in real world a bug report d_i may be handled by a number of people, only one of them, c_j ultimately resolves it—implements a bug fix or a requested feature, rejects a proposed enhancement, determines that the report is not really a bug, etc.—and therefore we assign to d_i the class label c_j .

Once we have built our model θ using the existing bug reports as training data, bug triage of a new bug report $d_{|\mathcal{D}|+1}$ simply follows from Equation 3: we assign it to the developer $c \in \mathcal{C}$ for whom $P(c|d_{|\mathcal{D}|+1}, \theta)$ is maximized.

4 Experimental results

To test the approach, we applied it to a selection of bug reports from the Eclipse project and tested its accuracy in assigning reports to developers.

4.1 Data set

We selected all reports entered into Eclipse’s bug tracking system⁶ between January 1, 2002 and September 1, 2002. A total of 15,859 reports were selected. The system records for each bug the id of the user⁷ who submitted it (the *submitter*), a one-line summary accompanied with a longer free-text description of the problem (which may include steps to reproduce it, or information from the error logs, core dumps, and stack traces), and various attributes such as its status (*new*, *resolved*, etc.), who it is assigned to, and the list of users on the “CC” list who are automatically notified of any changes to the report. The report can also

⁶Available online at <https://bugs.eclipse.org>

⁷“User” in this section denotes the user of the bug tracking system, who may be a developer actively working on the project, and occasional contributor, or simply a user of the software with interest in certain issues or features

contain a list of free-text comments which can be made by any user, and which include the author’s id and time of posting. Finally, each report stores a timestamped history of all the changes to its attributes, including the assigned-to.

To determine a document’s class (that is, the developer to whom it should be assigned), a straightforward approach would be to choose whoever was the report assigned to in the bug tracking system. However, this obvious approach is misleading for two reasons: first, in many cases this *assigned-to* “user is actually an email alias for a whole sub-team that deals with the module in question; second, just as often, the developer who actually implements the fix for the bug or requested feature—or who makes the decision to remove it from further consideration—is not the developer to whom the bug was nominally assigned in the bug tracking system.

Instead, we used our observations and experiences with the bug tracking and development procedures in the Eclipse project and devised the following heuristic to determine a report’s class (developer who should handle it from the out-set):

1. If the report was *resolved* by the assigned-to developer, the report is labelled by his or her class regardless of who the submitter was or what the report’s *resolution* was (e.g., *fixed*, *duplicate*, *invalid*, *later*, etc.). This is clearly the case of a developer who was in charge of the report and who has completed processing it.
2. If the report was *resolved* by someone other than the assigned-to developer, but not by the person who submitted it, we label the report with the class of the developer who marked it resolved. The reasoning is that whoever made the decision to resolve the report is the person to whom it should have been assigned all along.
3. If the report was *resolved* as *fixed*, regardless of who the resolver was, we assume that this is the developer who implemented the fix and label the report with the class of that developer, as this is probably the person who had done the real work on the report. This rule covers the frequent case where an Eclipse developer files a report, which is then assigned to somebody else or a sub-team alias by default, and then later implements the fix himself.
4. If the report was *resolved* as non-fixed (i.e., with resolution *duplicate*, *invalid*, etc.) by the person who submitted it, and who was not also assigned to it, the report is labelled with the class of the first person who responded to the reporter. This handles the many cases of a submitter throwing the report away after being informed that it is a feature and not a bug, or after being prompted by a developer for details of his or her setup and discovering that the bug does not exist any more.

We choose the first responder to the report rather than the assigned-to person for reasons outlined above.

5. If the report was *resolved* as non-fixed by the submitter who was not the assigned-to developer, and nobody responded, we assume that the report was submitted in error—for example, not knowing the proper operation of Eclipse—and that the mistake was caught by the submitter before anyone could react. These reports are removed from the training set, as they cannot be reliably labelled.
6. If the report was not *resolved*, we label it with the class of the most recent assigned-to developer.

These heuristics are not perfect, and we have noticed three or four examples where they are definitely not correct. However, based on a non-exhaustive examination of its results, they perform much better than always simply labelling a report with the class of the assigned-to developer. The labelling heuristics are obviously tailored to the development practices of the Eclipse project, and may need various amounts of modification before they could be applied to a different project. Mozilla, for example, uses a strict code review practice in which two senior developers need to check off on a proposed implementation (usually a patch that’s attached in the Bugzilla database), and so it is usually the reviewers who close the bug and not the developer responsible for the implementation.

During the labelling, we threw away 189 reports as described in step 4, for a total of 15,670 labelled documents (reports) and 162 classes (developers). We then extracted the summary and description of each report, tokenized all alphabetic sequences of characters (lower-cased and disregarding words in the standard SMART system stoplist of 524 common words such as “the”, “a”, etc.), and used that as the content of the document in classification. No stemming of words was done, except where so noted in the results section.

4.2 Methodology and measures

The data set was divided into a test set and a train set by randomly selecting a percentage of the documents from the data set for placing into the train set, with the remainder going to the test set. The model was learned using the train set, and then tested for label predictions of documents from the test set. We used the Bow toolkit [7], and configured it with the parameters as described above.

The classification accuracy was calculated as the percentage of documents for which the algorithm predicted the correct label. The predictions of course exist for all classes (that is, we calculate all the $P(c_j|d_i, \theta)$, where $\sum_j P(c_j|d_i, \theta) = 1$), but only the top prediction counts

when determining accuracy. The results reported below are the average over multiple runs, where each run used a new randomly built training and test sets (three runs per data point).

4.3 Results

In our experiments, we varied the size of the test set, the size of the vocabulary, and the criterion used to truncate the vocabulary. Figure 1 shows the classification accuracy as a function of the train/test set split, when the full vocabulary \mathcal{V} of words found in bug reports is used.

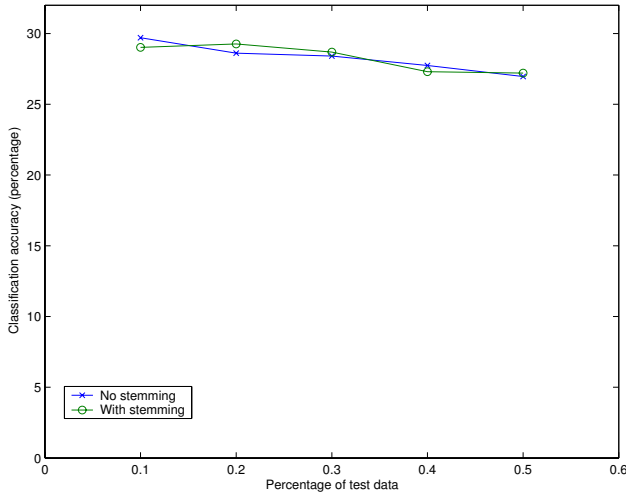


Figure 1. Classification accuracy without vocabulary truncation.

As we can see, the algorithm correctly assigns just under 30% of the bugs, when 90% of the document corpus is used as training and 10% as the test set. The accuracy slowly declines to 27% as the test set’s size is increased to 50% of the corpus.

Figure 1 also shows the results when the vocabulary was created using stemming, which identifies most grammatical variations of a word—such as “see,” “sees,” “seen,” for example—and treats them as a single term.⁸ The results are virtually unchanged, and any differences between the two conditions are within about one standard deviation at each data point.

Figure 2 shows the classification accuracy when the vocabulary was truncated to eliminate words that do not occur in at least d documents, for $d = \{1, 2, 5, 10, 20\}$. The accuracy is slightly lower than when the full vocabulary was used for $d = 1$, and almost a third worse (just above 20%) for $d = 20$. There is a slight downward trend as the size of the test set increases, but not in all cases.

⁸The standard Porter stemming algorithm was used.

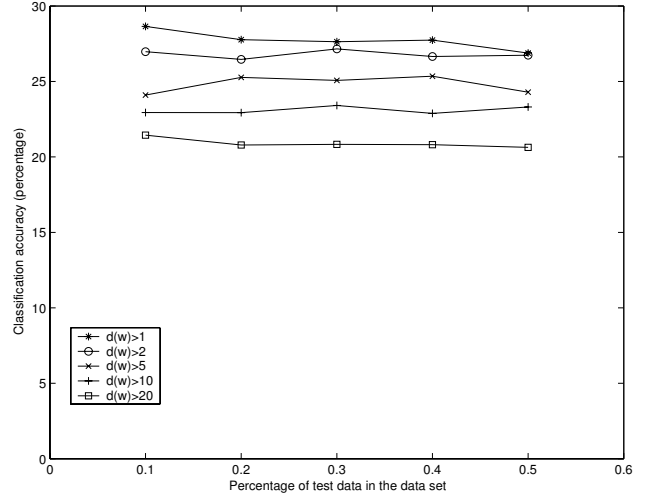


Figure 2. Classification accuracy when words occurring in fewer than d documents are removed from the vocabulary.

Figure 3 also shows the classification accuracy of a truncated vocabulary, but using a different truncating criterion. In this case, we eliminated all words that occur fewer than T times in the entire collection, for $T = \{5, 10, 20, 40, 80\}$. Again, the classification accuracy is slightly lower for $T = 5$ than when the full vocabulary is used, and falls to just over 20% for $T = 40$ and to around 18% for $T = 80$. Interestingly, for higher values of T , the accuracy improves with smaller training set, indicating some overfitting was occurring otherwise.

5 Discussion and future work

Overall, the performance of the algorithm was lower than expected, although the results are sufficiently promising to warrant further investigation. For example, we expected that truncating the vocabulary would have helped, by reducing the danger of overfitting, but that was clearly not the case, although smaller vocabulary speeds up the classification.

Also, we would like to involve the developers from the Eclipse project to evaluate the classification results based on their own subjective experience. For example, in those cases when a document is mis-classified, is the classification still “reasonable”—such as to a colleague on the same sub-team, who could handle the bug himself.

Our heuristics for deducing the developer-bug assignment in the data set could be improved further. It is currently based on our own observations of the bug-handling process, and could benefit from insight gained by directly involving the project’s developers. Also, we build our set of

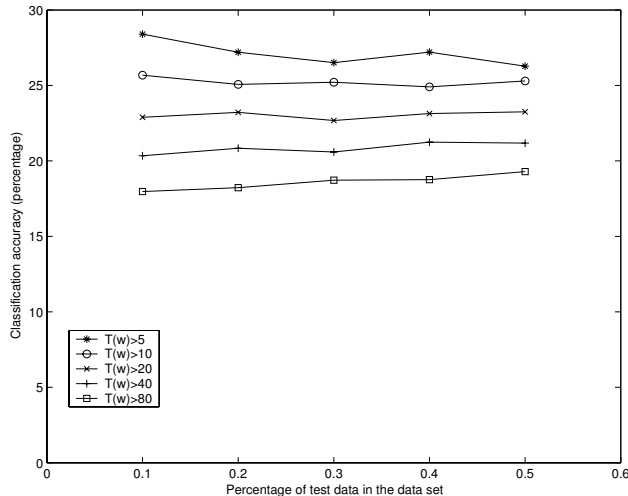


Figure 3. Classification accuracy when words occurring in the collection fewer than T times are removed from the vocabulary.

possible labels automatically from user ids in the bug tracking system. We could limit this set only to those users that we know are real developers on the team. This would eliminate labels corresponding to people who were perhaps only the bug submitters or interested bystanders, and who were falsely determined to be the bug “owners” by our heuristic.

Another weakness and a potential avenue of improvement is that in the process of creating the data set used in training and testing of the classifier, we either force a bug to a developer’s class, or throw it away. This is the consequence of the naive Bayes classifier algorithm that we use, which cannot deal with unlabelled documents in the corpus. However, there are extensions to this algorithm that combine it with Expectation Maximization (EM) methods to achieve very good results classifying a document corpus that contained a high proportion of unlabelled documents [8]. An interesting variation would be to label the documents with a range of probabilities, rather than just 1 or 0 we currently use, which would allow us to reflect our degree of certainty in the classification during the learning phase.

6 Summary

In this paper, we described an application of supervised machine learning using a naive Bayes classifier to automatically assign bug reports to developers. We evaluated our approach on bug reports from a large open-source project, Eclipse.org, achieving 30% classification accuracy with current prototype. We believe that the system could be easily incorporated into current bug-handling procedures to decrease the resources currently devoted to bug triage. New

bug reports would be automatically assigned to the developer predicted to be the most appropriate to the content. Mispredictions could be handled in a light-weight fashion by their assignee, “bouncing” them to a dedicated triager for human inspection and classification. Clearly, even the classification accuracy we can currently achieve, would significantly lighten the load that the triagers face under the present conditions.

References

- [1] I. T. Bowman and R. C. Holt. Reconstructing ownership architectures to help understand software systems. In *Proc. of IWPC 1999*, pp 28–37. IEEE Press.
- [2] T. Creasey. Personal communication, 14 July 2003. Eclipse developer, IBM Canada.
- [3] P. Domingos and M. Pazzani. Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In *Proc. of ICML 1996*, pp 105–112. Morgan Kaufmann.
- [4] M. Fischer, M. Pinzger, and H. Gall. Analyzing and relating bug report data for feature tracking. In *Proc. of WCRE 2003*, pp 90–99. IEEE Press.
- [5] T. Kalt. A new probabilistic method of text classification and retrieval. Technical Report IR-78, Center for Intelligent Information Retrieval, University of Massachusetts, Amherst, MA, 1996.
- [6] A. McCallum. Multi-label text classification with a mixture model trained by EM. In *AAAI Workshop on Text Learning*, 1999.
- [7] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, 1996.
- [8] K. Nigam, A. K. McCallum, S. Thrun, and T. M. Mitchell. Learning to classify text from labeled and unlabeled documents. In *Proc. of AAAI 1998*, pp 792–799. AAAI Press.
- [9] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated support for classifying software failure reports. In *Proc. of ICSE 2003*, pp 465–475. IEEE.
- [10] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1–2):69–90, 1999.
- [11] T. Zimmermann, P. Weißgerber, S. Diel, and A. Zeller. Mining version histories to guide software changes. In *Proc. of ICSE 2004*, to appear. IEEE Press.

Automatic Mapping of OWL Ontologies into Java

Aditya Kalyanpur

*Department of
Computer Science
University of
Maryland,
MD 20742 USA
aditya@cs.umd.edu*

Daniel Jiménez
Pastor

*Department of
Computer Science
University of
Bath, Bath BA2
7AY, UK
dani@pitaweb.com*

Steve Battle

*Hewlett Packard
Labs,
Bristol BS34 8QZ,
UK
steve.battle@hp.com*

Julian Padget

*Department of
Computer Science
University of
Bath, Bath BA2
7AY, UK
jap@cs.bath.ac.uk*

Abstract. We present an approach for mapping an OWL ontology into Java. The basic idea is to create a set of Java interfaces and classes from an OWL ontology such that an instance of a Java class represents an instance of a single class of the ontology with most of its properties, class-relationships and restriction-definitions maintained. We note that there exist some fundamental semantic differences between Description Logic (DL) and Object Oriented (OO) systems, primarily related to completeness and satisfiability. We present various ways in which we aim to minimize the impact of such differences, and show how to map a large part of the much richer OWL semantics into Java. Finally, we sketch the *HarmonIA* framework, which is used for the automatic generation of agent systems from institution specifications, and whose OWL Ontology Creation module was the basis for the tool presented in this paper.

1. Introduction

1.1 Fundamental Issues in Mapping

It must be stated from the outset there exist fundamental differences in understanding Description Logic based (DL) and Object-Oriented (OO) systems. These differences are inherent in the nature of the Knowledge Representation (KR) systems themselves and hence cannot be ignored.

For instance, a necessary and sufficient class definition in an ontology (DL-based system), which consists of restrictions on a set of properties, implies:

An individual which satisfies the property restrictions, belongs to the class

However, its equivalent class definition in Java (OO system) containing a set of fields with restrictions on field-values enforced through listener functions in its accessor methods implies:

A declared instance of the class is constrained by the field restrictions enforced through the class accessor methods

The above two definitions represent dual views of the same model, and hence are not semantically equivalent. However, we lay aside these differences with the hope that the mapping from DL to OO will serve its main purpose of aiding application development, as shown in the agent framework - *HarmonIA*.

2. Practical Benefits of Mapping OWL to Java

Grounding the abstract conceptual space of the ontological domain in the execution space of a programming language such as Java has many significant advantages. Primarily, the Java API generated from an ontology (schema) can be used to readily build applications (or agents) whose functionality is consistent with the design-stage specifications defined in the schema. Related work [1] by Andreas Eberhart amply demonstrates this concept. In the paper referenced, the author uses a tool called *OntoJava* to automatically develop a small link recommendation system (Java applet) called ‘*SmartGuide*’ from its Resource Description Framework Schema (RDFS) [2] specifications. We extend its applicability by focusing on the Web Ontology Language (OWL) [3], a more expressive logical formalism than RDFS, while additionally circumventing the need for a separate rule engine. Other benefits of this mapping include the use of any Java IDE to debug (or customize) the application or ontology easily and the use of *javadoc* to generate an online documentation of the ontology automatically.

3. Related Work

The concept of generating Java code from an ontology is not new. *OntoJava* (previously mentioned in section 2) is a cross compiler that translates ontologies written with *Protégé* [4] and rules written in RuleML [5] into a unified Java object database and rule engine. Similarly, the *Protégé* Bean Generator plug-in [6] can be used to generate FIPA/JADE compliant ontologies from RDF(S), XML and *Protégé* projects. A more comprehensive

solution, is the frame-based ontology language developed by Poggi *et al* [7], where a distinct ontology language inspired by KIF has been defined for use in the context of JADE, and from which the implementation of agent systems in JADE have been generated.

Our approach tries to learn from and build upon all the projects mentioned above (many of them designed to deliver Java implementations for Agent Systems), using the latest standards (OWL) from the W3C group. Furthermore, we intend to make our tool useful not only in Agent Systems, but also in a wider range of applications that are derived from their schema specifications. This is ensured by providing great engineering flexibility while building and debugging these applications through the use of listeners and specialized exceptions (as explained in section 4.2).

4. The Mappings in Detail

In order to build the instance of a Java class, we use the standard **Java-beans** [8] approach to access the values of the properties of the class (*set/get methods*). In order to maintain class relationships present in the ontology (including multiple inheritance), we use Java **interfaces** to define the OWL Classes. Finally, in order to enforce class-specific restrictions on properties, we use a set of *constraint-checker* classes that register themselves as **listeners** on the property inside the class definition, and are invoked upon property access to enforce the corresponding restriction.

4.1 OWL Classes

Every OWL class is mapped into a Java Interface containing just the accessor method declarations (*set/get methods*) for properties of that class (i.e. properties whose domain is specified as that class). Using an interface instead of a Java class to model an OWL class is the key to expressing the multiple inheritance properties of OWL, because Java's class language is single inheritance. In Java, an Interface can inherit from many others (unlike classes which can extend only a single class), and hence Interfaces are used in our mapping framework.

Because Java interfaces contain only static variables and abstract methods and are not by themselves instantiable, we define a corresponding Java class that embeds each interface (corresponding to an OWL class) wherein we explicitly define the fields (properties of the class) and implement the accessor methods. Additionally, based on the specified OWL property constraints, special-purpose *listeners* are registered on the corresponding field accessor methods within the Java class in order to enforce these constraints. Finally, every Interface inherits from *Thing*, an abstract Interface, needed to specify empty or unknown domains and ranges (explained in the next section).

OWL provides us with a set of complex operators (with explicit semantics) for defining classes such as *subClassOf*, *intersectionOf* and *oneOf*. Our approach maps complex OWL Class expressions to Java preserving the underlying semantics as much as possible. A summary of the mappings is shown in **Table 1**.

Note: For a more in-depth look at our mapping framework, with explanations for each type of mapping accompanied by UML diagrams and appropriate examples (with sample code), refer [9]

Table 1: OWL Class Mappings

	OWL	Java
Basic Class	A	interface IntA class A implements IntA
Class Axioms	$A \text{ equivalentClass } B$	interface IntAB extends IntA, IntB class A/B implements IntAB
	$B \text{ subClassOf } A$	interface IntB extends IntA
Class Descriptions	$A = \text{intersectionOf}(B, C)$	interface IntA extends IntB, IntC
	$A = \text{unionOf}(B, C)$	interface IntB/IntC extends IntA
	$A = \text{complementOf} / \text{disjointWith } B$	interface IntA { IntA ABBlocker(); interface IntB { IntB ABBlocker(); (Overridden blocking method – ABBlocker)
	$A = \text{oneOf}(I1, I2)$	Enum A{I1, I2}

4.2 OWL Properties

4.2.1 Domain

As mentioned earlier, all properties without a specified domain have their accessor functions declared in an abstract interface *Thing*, which is inherited by all Java Interfaces. However, if the domain of the property is specified as the ontology class X, the corresponding Java interface *IntX* contains declarations of accessor functions for the property. In the case of a multiple-domain property (net domain is an 'intersection-of' all the classes specified as the domain), we create an intersection interface (see **Table 1** for Java implementation of *intersection*) and declare accessor functions for the property inside this interface

4.2.2 Range

Unless stated otherwise (using appropriate restrictions), every property in OWL assumes multiple-cardinality, and hence the corresponding Java field must be of type *Collection*. Thus, OWL DatatypeProperties can be directly mapped into Java variables of the corresponding data type

collection (for e.g., properties with range xsd:String to fields of type String[] etc.), and ObjectProperties to Java variables whose type is the class specified in the property's range. However, while this mechanism works fine for single-range properties, it fails to account for multi-range properties as we now discuss.

In Java, each variable can be of only one type. This contrasts with the permitted multi-range properties in OWL. A possible workaround is to declare the variable of type *List*, which is a collection of generic objects, and to incorporate appropriate checking functions to ensure that only objects in the specified range are assigned to the variable. A natural technique to program this using the Java Beans API is to define a *RangeChecker* class for each multi-range property, which implements the *VetoableChangeListener* interface and that listens to the property change events. The *RangeChecker* class contains the *vetoableChange* method that is invoked each time the property value is set by using the *fireVetoableChange* function call inside the property's accessor function. The *vetoableChange* method then checks whether the type of the object being assigned to the property is in the valid range specified in its definition, if not, it vetoes the change throwing the corresponding exception. In our framework, we use a hierarchy of specialized exception classes such as *PropertyRangeException*, *FunctionalPropertyException*, *MinCardinalityException*, *HasValueException* etc, which are all (in some way) subclasses of the standard *PropertyVetoException* class.

To illustrate this concept, we consider a simple example. Say, we have an OWL ObjectProperty 'hasBrother' with its range defined as class – 'Male'. In our Java model, *hasBrother* is defined as a field of type *List*, with an associated *RangeMaleChecker* listener that listens to (and is invoked upon) any property change event. The *vetoableChange* method inside the *RangeMaleChecker* class contains a simple *if-then* statement to check whether object(s) in the *List* being assigned to the property are instances of class *Male* (using Java *instanceof*), if not, it vetoes the assignment and throws a *PropertyRangeException*.

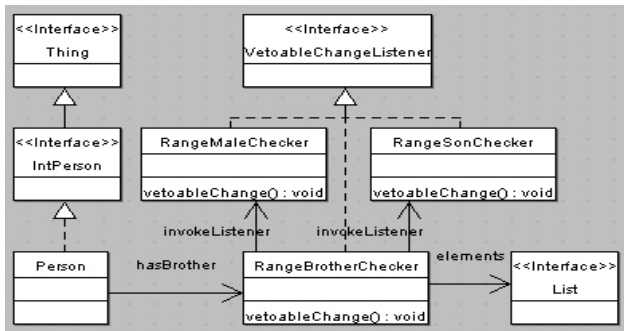


Figure 1: Multiple Range OWL Properties in Java

For properties with multiple-ranges, interpreted as an intersection of all ranges, the single-range mechanism can be directly extended as shown in **Figure 1**. Here, the range of the property *hasBrother* is defined as two separate classes, *Male* and *Son*. Hence the corresponding *RangeBrotherChecker* listener invokes the *RangeMaleChecker* and *RangeSonChecker* in turn to verify that an instance being assigned to the property satisfies both the type constraints.

This generic range-checking technique can easily handle single-range properties, or properties with multiple alternate ranges that are defined using the *unionOf* operator.

An added incentive for using listeners for property range checking or even cardinality/value checking is that it allows arbitrary extensions to the ontology, such as additional range axioms for a property, to be integrated seamlessly i.e. by just modifying code inside the *vetoableChange* method of the appropriate listener. Moreover, if desired, it provides the user with the flexibility of dynamically enforcing only a subset of the property constraints by turning on/off the appropriate listeners.

Table 2: OWL Property Mappings

	OWL	Java
Property Associations	<i>P domain A</i>	interface IntA {setP(List); List getP();}
	<i>Range</i>	VetoableChangeListener (range – instanceof, Functional – equal elements)
Property Descriptions	<i>Functional</i>	PropertyChangeListener (call appropriate accessor functions inside <i>propertyChanged()</i>)
	<i>InverseFunctional</i>	
	<i>Symmetric</i>	
	<i>Transitive</i>	
Property Relationships	<i>EquivalentProperty y</i>	VetoableChangeListener (check constraint satisfaction inside <i>vetoableChange()</i>)
	<i>SubPropertyOf</i>	
	<i>InverseOf</i>	
Property Restrictions	<i>Cardinality</i>	VetoableChangeListener (check constraint satisfaction inside <i>vetoableChange()</i>)
	<i>HasValue</i>	
	<i>SomeValuesFrom</i>	
	<i>AllValuesFrom</i>	

In addition to possessing basic attributes such as *domain* and *range*, OWL Properties can be defined using a set of reserved keywords such as *Functional*, *Symmetric* and *Transitive* that have predefined semantics. Moreover restrictions can be placed on the Property's value/cardinality (using operators such as *minCardinality* and *hasValue*) in order to define *Anonymous Classes*. Our

framework considers all the semantic nuances of OWL Property expressions while mapping to Java. A summary of the mappings is shown in **Table 2** (a detailed study is provided in [9]).

5. Tying it All Together

In order to demonstrate how the various components of the mapping framework fit together, we consider a simple, yet practical scenario – a domain ontology (in OWL) used to describe *Supply Chain Management* processes in a B2B environment.

Table 3: Fragment of a sample OWL Ontology

OWL Class	Definition / Description
<i>Company</i>	Primitive Class It's associated properties :- Datatype Properties who's domain is this Class and range is xsd:String include <i>businessName</i> , <i>contactAddress</i> , <i>contactPhone</i> , <i>businessID</i> (Functional) Object Properties who's domain is this Class and range is an appropriate OWL Class (can be guessed from the property's name) include <i>hasBusinessCategory</i> , <i>hasWorkOrder</i> , <i>hasPurchaseOrder</i>
<i>Automobile_</i> <i>Company</i>	subClassOf (<i>Company</i>) subClassOf (hasValue(<i>hasBusinessCategory</i> , "...#Automobiles"))
<i>Product</i>	Primitive Class It's associated properties :- Datatype Properties who's domain is this Class and range is xsd:String include <i>productName</i> , <i>productID</i> .. Object Property who's domain is this Class and range is <i>Company</i> is <i>hasMaker</i>
<i>Buyer</i>	intersectionOf (<i>Company</i> , ((≥ 1) <i>hasPurchaseOrder</i>))
<i>Purchase_Order</i>	subClassOf (<i>Order</i>) disjointWith (<i>Work_Order</i>)
<i>Automobile_Product</i>	subClassOf (<i>Product</i>) subClassOf (someValuesFrom(<i>hasMaker</i> , <i>Automobile_Company</i>))

Consider a subset of OWL Class definitions in the ontology as described in **Table 3**. The sample ontology contains concepts to represent *Company*, *Product*, *Purchase_Order* etc. that are integral components of a B2B supply-chain management process. Due to a lack of space, some of the related terms such as the concepts *Order* and *WorkOrder* are not displayed in Table 3. The ontology is rich in semantics, employing a suitable combination of OWL operators while defining specific concepts such as *Automobile_Company* and *Automobile_Product*.

We ground this OWL Ontology into Java using techniques described in [9] thereby illustrating the potential of our mapping framework in providing flexible execution capability while simultaneously preserving DL semantics. Space limitations restrict us from providing the complete Java code, however, **Table 4** contains some interesting fragments that we wish to discuss:

Table 4: Fragments of the equivalent Java Code

Java Element	Code
interface <i>IntProduct</i>	<pre>{.. void setProductName(List); List getProductName();.. void setHasMaker(List); List getHasMaker();.. ..}</pre>
interface <i>IntAutomobileProduct</i> extends <i>IntProduct</i>	<pre>{ // Java class implementing this interface registers the SVFAutomobileCompanyChecker Listener on property hasMaker }</pre>
class <i>SVFAutomobileCompany</i> <i>Checker</i> implements <i>VetoableChangeListener</i>	<pre>{.. void vetoableChange (PropertyChangeEvent evt) { List newValue = evt.getNewValue(); for (int i=0; i<newValue.size(); i++) if (newValue.elementAt(i) instanceof <i>AutomobileCompany</i>) return; throw SomeValuesFromException(); } ..}</pre>
interface <i>IntPurchaseOrder</i> extends <i>IntOrder</i>	<pre>{.. <i>PurchaseOrder</i> <i>PurchaseOrderWorkOrder</i> Blocker() {} ..}</pre>

5.1 Discussion

Observing **Table 4**, we see that Java interface *IntProduct* corresponds to the OWL Class *Product* and contains the appropriate accessor methods (set/get) for OWL Properties *productName*, *hasMaker* etc. These accessor methods accept and return arguments of type List. In Java 1.5, we can use generics to define these Lists of type *String* and *Company* respectively. **Note that we need to have a separate Java class embedding each interface in order to create usable Java objects (instances); in this case, Java class *Product* implements the *IntProduct* interface.**

On the same lines as *IntProduct*, we generate Java interface *IntCompany* (not shown in the table) corresponding to OWL Class *Company* with its corresponding property accessors declared within.

Now focusing on the interface *IntAutomobileProduct* defined in **Table 4**, we see that it corresponds to the OWL Class *Automobile_Product*. It extends the *IntProduct* interface thereby inheriting all its property accessor methods (maintaining the *subClassOf* relationship present in the ontology). Moreover, it needs to enforce the OWL restriction (*someValuesFrom*) on property *hasMaker*. For this, a special-purpose listener class called *SVFAutomobileCompanyChecker* is created. As shown in the table, this class implements the *VetoableChangeListener* interface and contains **constraint-checking** code inside the *vetoableChange* method. This code verifies that **there exists at least one element in the List** passed to the accessor method of the concerned property of type *AutomobileCompany*. This generic constraint checking method can handle other OWL restrictions such as *hasValue*, *allValuesFrom* etc. Moreover, this listener can also be registered on any other property that has the same restriction enforced in the ontology.

In this case, the listener *SVFAutomobileCompanyChecker* is registered on property *hasMaker* and fired from within its **set** accessor method inside the Java class *AutomobileProduct* that embeds the interface *IntAutomobileProduct*.

Note that an OWL restriction defines an anonymous class and we create intermediate Java interfaces to represent these anonymous OWL Classes. However, in the case of OWL Class *Automobile_Product*, since it's a subclass of OWL Class *Product*, and a restriction on property *hasMaker*, which has domain *Product*, there is no need to create an additional interface (corresponding to the anonymous class) containing the accessor methods for *hasMaker*, since these accessor methods are already declared in *IntProduct*. Our approach considers many optimization mechanisms such as this to keep the number

of Java interfaces and classes generated to a minimum (see [9]).

On the same lines as *IntAutomobileProduct*, we generate Java interface *IntBuyer* (not shown in the table) corresponding to OWL Class *Buyer* that is defined by the intersection of OWL Class *Company* and *minCardinality* restriction (≥ 1) on property *hasPurchaseOrder* (OWL-to-Java mappings for *intersectionOf* and *minCardinality* are similar to the mappings *subClassOf* and *someValuesFrom* respectively, refer [9]).

Finally, we focus on the interface *IntPurchaseOrder* defined in **Table 4** that corresponds to OWL Class *Purchase_Order*. Since its ontological definition has it *disjointWith* OWL Class *Work_Order*, we declare two identically named blocking functions *PurchaseOrderWorkOrderBlocker()* having different return types in each of the respective Java interfaces – *IntPurchaseOrder* and *IntWorkOrder* (latter not shown in the table). In this case, the return type of the blocking method is the corresponding Java class name. Thus, a third interface cannot extend both – *IntPurchaseOrder* and *IntWorkOrder* together without incurring a compiler error, thereby preserving its disjoint semantics.

Thus, to summarize, we have shown how our proposed OWL-to-Java mapping approach maps simple ontological axioms, as well as more complex expressions built using basic 'building block' axioms, into Java, preserving the DL semantics as much as possible.

Now imagine an agent-toolkit that uses the Java API generated from the ontology. It can create instances of Java Classes *Company*, *Product* etc, and use its accessor functions to populate the instance, at each point, ensuring validity and consistency of the ontological model. In a multi-agent environment sharing a common global ontology, individual agents can communicate with each other seamlessly by interchanging Java Objects (instances) of appropriate Java classes (corresponding to the respective OWL classes), maintaining semantic integrity at all times.

It must be emphasized that given the nature of our framework, very large ontologies (containing >500 classes) cannot be mapped directly into Java without being broken down into smaller, more manageable ontologies. However, given that most domain ontologies in today's software systems (esp. agent-based systems) are small-to-moderately sized, our framework should suffice without significant modification.

6. Testbed Application: *HarmonIA*

HarmonIA is a framework for the automatic generation of e-organizations from institution specifications, created at the University of Bath. Its *Ontology Creator* module, developed by Daniel Jiménez Pastor, generates

FIPA/JADE compliant ontologies from OWL ontology specifications using Jena 2 [10], thus serves as a natural basis for a tool that implements the work covered in this paper. A preliminary report on the framework appeared in [11]. This paper is a refinement and extension of the ontology technology, and a comprehensive discussion of the *HarmonIA* toolset is in preparation.

The *Ontology Creator* module in *HarmonIA* is being extended to handle ontological features such as multiple-inheritance among classes, multiple-range properties etc that Java does not support. Our current goal is to complete the *Ontology Creator* module by implementing all the features presented in this paper, thereby providing an automated CASE tool for the mapping of OWL into Java. This will be released open-source in the form of an API, which can be easily embedded into third party applications.

7. Future Work

Our future goal is to design a Protégé plug-in based on our OWL-to-Java mappings, which together with the recently released OWL plug-in [12] would help provide a complete graphical framework for creating OWL ontologies and obtaining the equivalent Java UML class representation.

Finally, we wish to further enhance our mapping framework (by possibly borrowing techniques described in [1]) to include advanced property definitions (*owl:InverseFunctionalProperty*), axioms relating individuals (*owl:sameAs* etc) and OWL rules [13], thereby providing for sophisticated A-Box reasoning.

8. Conclusion

We have presented a concise and elegant solution to map an OWL Ontology into the Java Language. OWL has three sub-languages (OWL Lite, OWL DL and OWL Full) and we attempt to deliver a solution for the most expressive - OWL Full - at the expense of completeness, which is inevitably compromised in migrating to the OO domain. Certain OWL constructs (that primarily fall into A-Box reasoning) have not been accounted for, since incorporating their semantics is either forbidden within the Java framework, or requires a significant amount of additional code. However, ignoring them does not reduce the main utility of our mappings, which is aiding systematic application development, as was exemplified by the *HarmonIA* system.

We hope that more research continues in this direction in order to realize practical widely used applications based on Semantic Web technologies.

9. References

[1] Andreas Eberhart, "Automatic Generation of Java/SQL based Inference Engines from RDF Schema and RuleML". I. Horrocks

and J. Hendler, editors, Proceedings of the First International Semantic Web Conference (ISWC 2002), Chia, Sardinia, Italy, pages 102--116, June 2002

[2] Resource Description Framework Schema (RDFS) Specifications by the W3C. <http://www.w3.org/TR/rdf-schema/>

[3] Word Wide Web Consortium (W3C). OWL – Web Ontology Language. <http://www.w3.org/TR/owl-ref>

[4] Protégé, an editor for ontologies (software package) <http://protege.stanford.edu>

[5] The Rule Markup Initiative: <http://www.dfki.uni-kl.de/ruleml/>

[6] Bean Generator plug-in for Protégé: <http://gaper.wi.psy.uva.nl/beangenerator>

[7] Poggi, F. Bergenti, and F. Bellifemine. "An ontology description language for FIPA agent systems". Technical Report DII-CE-TR001-99, University of Parma, 1999.

[8] Java Beans API: <http://java.sun.com/products/javabeans/>

[9] Aditya Kalyanpur et al, Detailed Technical Report on Mapping OWL to Java http://www.mindswap.org/~aditkal/OWLtoJava_Report.pdf

[10] Jena, HP Labs Semantic Web Toolkit. <http://jena.sourceforge.net/>

[11] D. Jiménez Pastor and J. Padget. "Towards HarmonIA: automatic generation of e-organisations from institution specifications". Workshop on Ontologies in Agent Systems (OAS'03, <http://oas.otago.ac.nz/OAS2003>), July 2003, Melbourne, Australia.

[12] OWL Plug-in for Protégé <http://protege.stanford.edu/plugins/owl/>

[13] I. Horrocks, P. P. Schneider, H. Boley, S. Tabet, "A Proposal for an OWL Rules Language": <http://www.daml.org/rules/proposal/>

Black- and White-Box Self-testing COTS Components*

Sami Beydeda and Volker Gruhn
University of Leipzig
Chair of Applied Telematics / e-Business
Klostergasse 3
04109 Leipzig, Germany
{beydeda,gruhn}@ebus.informatik.uni-leipzig.de

Abstract

Development of a software system from existing components can surely have various benefits, but can also entail a series of problems. One type of problems is caused by a limited exchange of information between the developer and user of a component, i.e. the developer of a component-based system. A limited exchange of information cannot only require the testing by the user but it can also complicate this tasks, since vital artifacts, source code in particular, might not be available. Self-testing components can be one response in such situation. This paper describes an enhancement of the Self-Testing COTS Components (STECC) Method so that an appropriately enabled component is not only capable of white-box testing its methods but also capable of black-box testing.

1 Introduction

Quality assurance, including testing, conducted in development and use of a component can be considered according to [12, 11] from two distinct perspectives. These perspectives are those of the *component provider* and *component user*. The component provider corresponds to the role of the developer of a component and the component user to that of a client of the component provider, thus to that of the developer of a system using the component.

The use of components in the development of software systems can surely have several benefits, but can also introduce new problems. Such problems concern, for instance, testing of components. The component user has often to test a component, particularly a third-party component, prior to its integration into the system to be developed. The various reasons obligating the component's testing by the compo-

nent user are outlined in [7] with an overview of existing approaches to testing components.

In this paper, we describe an enhancement of the *Self-Testing COTS Components (STECC) Method* [4, 6]. The main idea of the STECC method is to augment a component with self-testability, so that the component user can test it thoroughly without necessitating the component provider to disclose certain information. In particular, a STECC self-testing component allows white-box tests without access to the component's source code. Source code information is processed within the component in an encapsulated manner not visible to the component user.

The enhancement of the STECC method addresses the need that the component user often not only needs to white-box test the component, but also black-box test according to the component's specification. For this purpose, the internal model encapsulated in a STECC self-testing component, which is particularly used for test case generation, has been augmented to also embrace information extracted from its specification. We, however, have not developed a new model, but rather use one which reached a certain maturity in testing classes, the *Class Implementation Specification Graph (CSIG)* [8]. Note that in the following a component is assumed to be implemented as a class, such as components according to the Enterprise JavaBeans Specification [9]. A positive side effect of CSIGs is that they do not only allow an integrated black- and white-box testing, the total number of test cases required for black- and white-box testing can be less than in the case when both tasks are carried out separately [8].

2 Self-Testing COTS Components Strategy

The component provider and component user generally need to exchange information during the various phases of developing the component and a component-based system [6]. Various factors, however, impact the exchange of

*The chair of Applied Telematics / e-Business is endowed by Deutsche Telekom AG.

information between the component provider and component user. The information requested by one role and delivered by the other can differ in various aspects, if it is delivered at all. It can differ syntactically insofar that it is, for instance, delivered in the wrong representation and it can also differ semantically in that it, for instance, is not in the abstraction level needed.

A lack of information might require the testing of a component by its user prior to its integration in a system, and might significantly complicate this task at the same time. The component user might not possess the information required for this task. Theoretically, the component user can test a component by making certain assumptions and approximating the information required. Such assumptions, however, are often too imprecise to be useful. For instance, control-dependence information can be approximated in safe-critical application contexts by conservatively assuming that every component raises an exception, which is obviously too imprecise and entails a higher testing effort than necessary [12, 11].

Even though often claimed, source code as one type of information often required for testing purposes is not required by itself for testing purposes. It often acts as the source for obtaining other information, such as that concerning control-dependence. Instead making source code available to allow the generation of such information, the information required can also be directly delivered to the component user, obviating source code access. This type of information is often referred to as *meta-information* [17]. Even though the information required might already be available from own testing activities, the component provider might nevertheless not deliver this information to the component user. One reason may be that detailed information, including parts of the source code, can be deduced from it depending on the granularity of the meta-information. Therefore, there is a natural boundary limiting the level of detail of the information deliverable to the user. For some application contexts, however, the level of detail might be insufficient and the component user might not be able to test the component according to certain quality requirements.

The underlying strategy of the method proposed differs from those discussed thus far. Instead of providing the component user with information required for testing, component user tests are supported by the component explicitly. The underlying strategy of the method is to augment a component with functionality specific to testing tools. A component possessing such functionality is capable of testing its own methods by conducting some or all activities of the component user's testing processes, it is thus *self-testing*. The method is thereby called the *Self-Testing COTS Components* (STECC) method. Self-testability does not obviate the generation of detailed technical information. In fact,

this information is generated by the component itself during runtime and is internally used in an encapsulated manner. The information generated is transparent to the component user and can thus be more detailed than in the case above. Consequently, tests carried out by the component user through the self-testing capability can thereby be more thorough as in the case of meta-information. Self-testability allows the component user to conduct tests and does not require the component provider to disclose source code or other detailed technical information. It thereby meets the demands of both parties. The STECC method does not only benefit the user of a component in that the user can test a component as required. It can also benefit its provider, as self-testability provided by an appropriately augmented component can be an advantage in competition.

From a technical point of view, a STECC self-testing component maintains a model of its own and generates test cases with regards to an adequacy criterion specified by the tester, who can particularly be the component user. The STECC framework, which implements the various relevant algorithms, determines the paths to be traversed according to the specified criterion and generates the necessary test cases as possible. The test case generation algorithm employed for this purpose is the *Binary Search-based Test Case Generation (BINTEST) Algorithm* [5]. The internal model used by the component is a control flow graph. It can be replaced by another control flow graph as long as its syntactical representation does not change. This is exactly the enhancement of the STECC method described in this paper. CSIGs are control flow graphs which also embrace specification information and thus allow generation of black-box test cases.

3 Class Implementation Specification Graphs

3.1 Motivation

Analysis and testing tasks are usually conducted using a model of the program under consideration which abstracts from certain aspects and focuses on others assumed to be more significant. Typical examples of such models are control flow graphs or finite state machines. Models used in analysis and testing are often constructed on the basis of the implementation, such as control flow graphs, or the specification, such as finite state machines, of the program under consideration, they seldom cover both. However, we often need to analyze and test a program according to both information sources. In the case of class-level analysis and testing, one answer to this need are *Class Specification Implementation Graphs* (CSIGs) [8].

The distinguishing feature of CSIGs from existing class models is that they combine the specification and imple-

mentation of a class. Each method is represented by two control flow graphs in possibly different abstraction levels, i.e. control flow as specified and control flow as implemented. We refer to the former as the *specification view* and the latter as the *implementation view* of a method. Therefore, this model is called the *class specification implementation graph* (CSIG) of a class to emphasize the combination of the two different views. Although the method views can differ in abstraction level, the difference does not affect the integration, as the integration is carried out at control flow graph level. As control flow graphs are used to model specification and implementation, structural techniques, such as the BINTEST algorithm, can be used for test case generation. An important feature of a CSIG is that generated test cases can cover both specification and implementation.

3.2 A demonstrative example

The example consists of a component, called *account*, which simulates a bank account. This component provides the appropriate methods for making bank account deposits (*deposit()*) and withdrawals (*withdraw()*). Furthermore, it provides methods for paying interest (*payInterest()*) and for printing bank statements (*printBankStatement()*).

Figure 1 shows the specification of component *account* in form of a *class state machine* (CSM) [14]. In this figure each state of component *account* is represented by a circle, while each transition is depicted by an arrow leading from its source state to its target state. These transitions are formally specified through 5-tuples (*source, target, event, guard, action*) below this figure. A transition consists – besides a *source* and a *target* state – of an *event* causing the transition, a predicate *guard* which has to be fulfilled before the transition can occur, and an *action* defining operations on the attributes during the transition. There are also two special circles labeled *initial* and *final*. These two circles represent the state of a component before its instantiation and after its destruction, respectively. Thus, they represent states in which the attributes and their values are not defined, meaning that these two states are not concrete states of a component instance. For the sake of brevity, below the CSM in figure 1 only the transitions with event type *deposit()* are given.

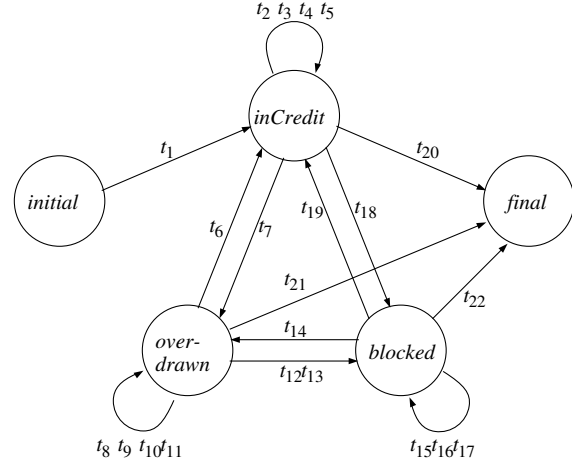
A possible implementation of method *deposit()* is:

```

33 public void deposit(double amount) {
34     balance += amount;
35     t[idx++] = new transaction("Deposit ",
36                             amount, balance);
37 }

```

With *transaction* implementing a financial transaction stored in an array *t* for later generation of bank statements.



inCredit := *balance* >= 0

overdrawn := *balance* < 0 && *balance* >= *limit*

blocked := *balance* < *limit*

*t*₂ = (*inCredit*, *inCredit*, *deposit*(*amount*), *true*, *balance* += *amount*;)
*t*₇ = (*overdrawn*, *inCredit*, *deposit*(*amount*), *balance*+*amount* >= 0, *balance* += *amount*;)
*t*₈ = (*overdrawn*, *overdrawn*, *deposit*(*amount*), *balance*+*amount* < 0, *balance* += *amount*;)
*t*₁₄ = (*blocked*, *overdrawn*, *deposit*(*amount*), *limit* <= *balance*+*amount* && *balance*+*amount* < 0, *balance* += *amount*;)
*t*₁₅ = (*blocked*, *blocked*, *deposit*(*amount*), *balance*+*amount* < *limit*, *balance* += *amount*;)
*t*₁₉ = (*blocked*, *inCredit*, *deposit*(*amount*), *balance*+*amount* >= 0, *balance* += *amount*;)

Figure 1. Specification of component *account* by a class state machine

3.3 CSIG constituents

Figure 2 shows the CSIG of component *account*. Each method of a component is represented by two control flow graphs in its CSIG. One of them is a control flow graph generated on the basis of the method specification (*method specification graph*), whereas the other is a control flow graph determined using the method implementation (*method implementation graph*). In figure 2, method specification graphs are drawn light gray whereas method implementation graphs are drawn dark gray. For convenience, the two control flow graphs are called *method graphs*, if they do not have to be distinguished. Thus, the CSIG of a component shows each method from two different perspectives, namely what the method should do and what the method actually does.

The two method graphs of each method are embedded within a frame structure called a *class control flow graph frame* (CCFG frame). Generally, a component cannot be tested without a test driver, which creates an instance of the

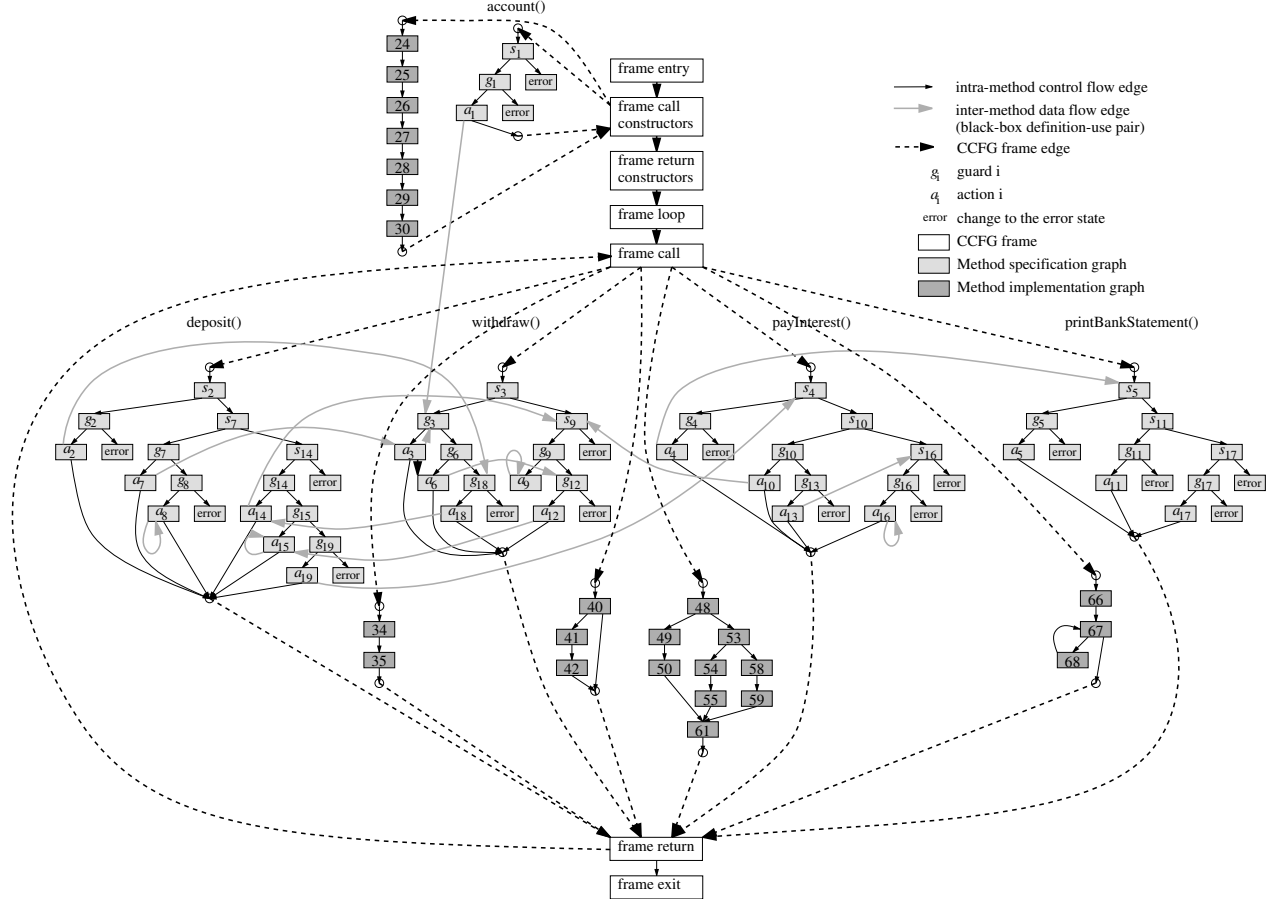


Figure 2. Class specification implementation graph of component account

component, invokes the corresponding methods in a particular order, and finally deletes the instance. A CCFG frame represents an abstract test driver fulfilling this task¹. In figure 2, the CCFG frame nodes are drawn without shading.

Three types of edges can be distinguished within a CSIG:

1. *Intra-method control and data flow edges*

Intra-method control and data flow edges depict control and data dependencies within a single method graph. For instance, an intra-method data flow edge connects a node representing a definition of a local variable with another node representing a use in the same method (as a simple example of a def-use pair). In figure 2, these edges are drawn as solid arrows.

2. *Inter-method control and data flow edges*

Edges of this type model control and data flow between two method specification graphs and two method implementation graphs, respectively. Assume that G_1

and G_2 are the method implementation graphs of methods M_1 and M_2 , respectively. Then, an invocation of method M_2 within the implementation of method M_1 is modeled by an inter-method control flow edge leading from the corresponding node in G_1 to the entry node of G_2 . In figure 2, these edges are shown as gray arrows. For the sake of simplicity, this type of edges is only given for method specification graphs.

3. *CCFG frame edges*

The third type of edges in a CSIG consists of nodes, which either connect two CCFG frame nodes or the CCFG frame with entry and exit nodes of method graphs. In figure 2, this type of edges is shown as dotted arrows.

Method implementation graphs are generated on the basis of the implementations of the respective methods. Control flow graph generation is, for instance, described in [1]. The generation of method specification graphs is conducted on the basis of *method prototypes*, which are constructed us-

¹A CCFG frame is a part of a CCFG suggested by Harrold et al. [13] for class-level data flow testing. As we only need the frame structure as an abstract test driver, we do not introduce CCFGs in this paper.

ing the finite state machine specification of the component.

For the construction of method prototypes, a prototype is generated for each transition $t = (source, target, event, guard, action)$ in the form of a nested if-then-else construct:

```
if (source)
  if (guard)
    action;
  else throw new ErrorStateException();
else throw new ErrorStateException();
```

source refers to the predicate of the source state. For instance, the predicate of state *inCredit* is defined as $balance \geq 0$.

After generating these prototypes, those having the same event type are combined. For instance, transitions $t_2, t_7, t_8, t_{14}, t_{15}$ and t_{19} share the event `deposit()`. Their prototypes can be merged to the following method prototype:

```
deposit(double amount) {
s2  if (balance >= 0)
g2  if (true)
a2  balance += amount;
    else throw new ErrorStateException();
    else
s7  if (balance < 0 && balance >= limit)
g7  if (balance + amount >= 0)
a7  balance += amount;
    else
g8  if (balance + amount < 0)
a8  balance += amount;
    else throw new ErrorStateException();
    else
s14 if (balance < limit)
g14 if (limit <= balance + amount
    && balance + amount < 0)
a14 balance += amount;
    else
g15 if (balance + amount < limit)
a15 balance += amount;
    else
g19 if (balance + amount <= 0)
a19 balance += amount;
    else throw new ErrorStateException();
    else throw new ErrorStateException();
}
```

Generation of control flow graphs for method prototypes can again be carried out as described in [1]. The process of embedding the various control flow graphs into a CCFG frame is explained in [13].

In the STECC method as initially designed, a component encapsulates an ordinary control flow graph modeling source code information. Tests as conducted by a STECC self-testing component were therefore solely white-box oriented. An enhancement of the STECC method to also cover black-box tests can be achieved by using CSIGs instead of ordinary control flow graphs. CSIGs also model specification information and tests conducted are thus also black-box oriented. Specifically, the total number of test cases required can even be less than in the case when black-box and white-box testing separately. A suitable test suite reduction technique is described in [8].

4 Related work

The STECC approach can be compared to built-in testing approaches in the literature. A number of built-in testing

approaches have been proposed in the literature, e.g. [19], [16, 18, 10, 3] and [15, 2]. Similar as the STECC approach, built-in testing approaches aim at tackling difficulties in testing components caused by a lack of information, difficulties in test case generation in particular. The STECC approach has the same objective and the approaches can thus be directly compared to it. A comparison of them highlights several differences.

Firstly, the built-in testing approaches are static in that the component user cannot influence the test cases employed in testing. A component which is built-in testing enabled according to one of these approaches either contains a predetermined set of test cases or the generation, even if conducted on-demand during runtime, solely depends on parameters which the component user cannot influence. However, the component user might wish to test all components to be assembled with respect to an unique adequacy criterion. Built-in testing approaches usually do not allow this. The STECC approach does not have such a restriction. Adequacy criteria, even though constrained to control flow criteria, can be freely specified.

Secondly, built-in testing approaches using a predefined test case set generally require more storage than the STECC approach. Specifically, large components with high inherent complexity might require a large set of test cases for their testing. A large set of test cases obviously requires a substantial amount of storage which, however, can be difficult to provide taking into account the storage required in addition for execution of large components. This is also the case if test cases are stored separately from the component, such as proposed by component+ approach. In contrast, the STECC strategy does not require predetermined test cases and does also not store the generated test case.

Thirdly, built-in testing approaches using a predefined test case set generally require less computation time at component user site. In such a case, the computations for test case generation were already conducted by the component provider and obviously do not have to be repeated by the component user, who thus can save resources, particularly computation time, during testing. Savings in computation time are even magnified if the component user needs to frequently conduct tests, for instance, due to volatility of the technical environment of the component. Storage and computation time consumption of a built-in testing enable component obviously depends on the implementation of the corresponding capabilities and the component provider needs to decide between the two forms of implementation, predefined test case set or generation on-demand, carefully in order to ensure a reasonable trade-off.

Fourthly, none of the existing built-in testing approaches, at least those known to the authors, are capable of providing or generating test cases for both black- and white-box testing. This is to our opinion the most significant difference.

5 Conclusions

The STECC strategy addresses the needs of both the component provider and component user. A situation particularly encountered in the case of commercial components, thus COTS components, is that the component provider might not wish to disclose information, particularly source code, which the component user might require for testing purposes. Our research started with the observation that existing approaches do not appropriately tackle such a situation.

The STECC strategy is a response to such situations. It allows the component user to test the component and to ensure suitability of the component to the target application context regarding its quality without requiring the component provider to publish specific information. It thus meets the demands of both parties. The STECC strategy can lead to a win-win situation insofar that both the component provider and component user can benefit from it. The benefit of the component user is obvious. The STECC strategy, or more clearly self-testability of a component, can be a valuable factor in competition. This potential benefit of the component provider from the STECC strategy becomes more obvious taking into account the specific type of components which are the most appropriate candidates for STECC self-testability, COTS components.

We have shown that an enhancement of the STECC method is easily possible using CSIGs. CSIGs represent both specification and implementation information and tests conducted are thus black- and white-box oriented. Specifically, the total number of test cases required can even be less than in the case when black-box and white-box testing separately.

References

- [1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, principles, techniques, and tools*. Addison Wesley, 1988.
- [2] C. Atkinson and H.-G. Groß. Built-in contract testing in model-driven, component-based development. In *ICSR Workshop on Component-Based Development Processes*, 2002.
- [3] B. Baudry, V. L. Hanh, J.-M. Jezequel, and Y. L. Traon. Trustable components: Yet another mutation-based approach. In W. E. Wong, editor, *Mutation testing for the new century*, pages 47–54. Kluwer Academic Publishers, 2001.
- [4] S. Beydeda. *The Self-Testing COTS Components (STECC) Method*. ISBN 3-89975-462-X, Martin Meidenbauer Verlag, München, 2004.
- [5] S. Beydeda and V. Gruhn. BINTEST - binary search-based test case generation. In *Computer Software and Applications Conference (COMPSAC)*, pages 28–33. IEEE Computer Society Press, 2003.
- [6] S. Beydeda and V. Gruhn. Merging components and testing tools: The self-testing COTS components (STECC) strategy. In *EUROMICRO Conference - Component-based Software Engineering Track (EUROMICRO)*, pages 107–114. IEEE Computer Society Press, 2003.
- [7] S. Beydeda and V. Gruhn. State of the art in testing components. In *International Conference on Quality Software (QSIC)*, pages 146–153. IEEE Computer Society Press, 2003.
- [8] S. Beydeda, V. Gruhn, and M. Stachorski. A graphical representation of classes for integrated black- and white-box testing. In *International Conference on Software Maintenance (ICSM)*, pages 706–715. IEEE Computer Society Press, 2001.
- [9] L. G. DeMichiel. Enterprise javabeans specification, version 2.1. Technical report, Sun Microsystems, 2002.
- [10] D. Deveaux, P. Frison, and J.-M. Jezequel. Increase software trustability with self-testable classes in java. In *Australian Software Engineering Conference (ASWEC)*, pages 3–11. IEEE Computer Society Press, 2001.
- [11] M. J. Harrold. Testing: A roadmap. In *The Future of Software Engineering (special volume of the proceedings of the International Conference on Software Engineering (ICSE))*, pages 63–72. ACM Press, 2000.
- [12] M. J. Harrold, D. Liang, and S. Sinha. An approach to analyzing and testing component-based systems. In *International ICSE Workshop Testing Distributed Component-Based Systems*, 1999.
- [13] M. J. Harrold and G. Rothermel. Performing dataflow testing on classes. In *Second ACM SIGSOFT Symposium on the Foundations of Software Engineering (New Orleans, USA)*, volume 19 of *ACM SIGSOFT Software Engineering Notes*, pages 154–163. ACM Press, 1994.
- [14] H. S. Hong, Y. R. Kwon, and S. D. Cha. Testing of object-oriented programs based on finite state machines. In *Second Asia-Pacific Software Engineering Conference (Brisbane, Australia)*, pages 234–241. IEEE Computer Society Press, 1995.
- [15] J. Hörnstein and H. Edler. Test reuse in cbse using built-in tests. In *Workshop on Component-based Software Engineering, Composing systems from components*, 2002.
- [16] J.-M. Jezequel, D. Deveaux, and Y. L. Traon. Reliable objects: Lightweight testing for oo languages. *IEEE Software*, 18(4):76–83, 2001.
- [17] A. Orso, M. J. Harrold, and D. Rosenblum. Component metadata for software engineering tasks. In *International Workshop on Engineering Distributed Objects (EDO)*, volume 1999 of *LNCS*, pages 129–144. Springer Verlag, 2000.
- [18] Y. L. Traon, D. Deveaux, and J.-M. Jezequel. Self-testable components: from pragmatic tests to design-to-testability methodology. In *Technology of Object-Oriented Languages and Systems (TOOLS)*, pages 96–107. IEEE Computer Society Press, 1999.
- [19] Y. Wang, G. King, and H. Wickburg. A method for built-in tests in component-based software maintenance. In *European Conference on Software Maintenance and Reengineering (CSMR)*, pages 186–189. IEEE Computer Society Press, 1999.

Building on-line sales assistance systems with ADVISOR SUITE

Dietmar Jannach and Gerold Kreutler

Institute for Business Informatics and Application Systems, University Klagenfurt
{dietmar,gerold}@ifit.uni-klu.ac.at

Abstract. Online sales advisory systems guide customers through the decision and buying process and provide added value for online customers, especially in domains where a wide range of comparable products is available and differentiation between the products requires deep technical knowledge. In this paper we present ADVISOR SUITE, an expert system that combines knowledge-based approaches for personalized product recommendation with an adaptive user interface, which is used to guide the customer through the requirements elicitation process according to his personal needs and preferences. A particular focus of the presented system lies on the reduction of costly knowledge engineering and maintenance times, which is addressed by a consistent set of graphical tools, and a personalization approach that enables the automatic generation of large parts of the user interface of the advisory application.

1. Introduction

In today's competitive markets, customers can choose among multiple suppliers for the desired goods whereby in many cases the differences between the products are only understandable for an expert with deep technical knowledge. In an electronic shop, customers can, for instance, choose among several hundreds of digital cameras; clients of a bank have a choice of hundreds of different funds or other forms of investments. In the classical sales channel, an experienced sales assistant will query his customer about his preferences and needs. Consequently, he will adapt the communication style in this dialogue depending on, e.g., the customer's (technical) knowledge or interests and then use his domain knowledge to propose a ranked list of suitable product alternatives [12, 1]. At the end of the dialogue, the sales person will give the customer an explanation for his recommendation and provide additional hints. In the online-buying channel, however, there has only been little support in that area for a long time and only in recent years the importance of the added-value of online sales assistance was recognized and suitable methods were developed [18], e.g., personalized recommender systems based on collaborative filtering [13, 16], data mining techniques, or

expert systems based on decision trees or case-based reasoning techniques [14, 19, 3, 4].

The ADVISOR SUITE framework presented in this paper is a knowledge-based approach for building sales assistance systems in arbitrary domains that simulate the behaviour of an experienced sales person. Beside dialogue-based requirements elicitation and constraint-based product selection, the system also comprises a framework for rapid development of adaptive and personalized user interfaces that adapt themselves to the knowledge level of the current customer [1]. In order to reduce the typically high development and maintenance costs for such knowledge-intensive expert system applications, we developed a consistent set of graphical and intuitive knowledge engineering tools. Our experiences show that these tools can be used by the domain experts themselves after a short training phase which significantly reduces development and maintenance times for the applications.

The paper is organized as follows. After the presentation of the novel constraint-based approach for personalized product selection and ranking, we discuss how advisory dialogues can be modeled with ADVISOR SUITE and how this knowledge is then exploited by a generic and parameterizable user interface component. Then, we present the overall architecture of the system, make a comparison with related work in the field and end with a discussion of practical experiences of several industrial applications.

2. Recommendation Techniques

For determining the set of suitable products for the customer ("filtering"), we apply a constraint-based approach which exploits explicit knowledge about product features and customer requirements. Thus, it is possible to represent the domain knowledge of an experienced sales person in a declarative knowledge base. The structure of the knowledge base can be sketched as follows. Products are characterized by a fixed set of *attributes*, whereby these attributes take one or several values from a pre-specified domain (like numbers, text, or predefined enumerations). Note, that in particular *multi-valued attributes* are common in practical settings, e.g., a digital camera can support multiple image formats. *Customer properties*

represent the current user's characteristics, skills and interests. The domain expert defines a set of *questions* (and possible answers) that he would pose in order to acquire the customer preferences. Together, product attributes and customer properties define the variables in the filtering mechanism. The expert rules for recommendations are denoted as *filter constraints*; typical examples for such rules within the financial domain are, for instance:

- If the customer is not interested in emerging markets or can only take low risks, we would propose conservative investments.
- In any case, only propose products where the monthly payments correspond to the customer's preferences plus/minus 10 percent.
- If the customer responded with "yes" to questions A and B, only recommend long-term investments.

The system's language for describing the expert rules therefore includes arithmetic and relational operators on customer properties and product attributes, text manipulation, if-then-else style constructs, as well as set operations for multi-valued attributes. The knowledge acquisition process for the expert rules is supported by a graphical user interface (see Figure 1) with online input assistance and a simplified user-oriented notation.

Besides explicitly questioning a customer about his preferences, the proposed framework supports the derivation of additional customer properties based on indirect questions. In particular, this is helpful in situations where the end-user of the system is no domain expert [1]. As an example, an investment advisor will ask several questions about the customer's financial or family situation in order to determine the suitable risk class of investments.

One of the major criticisms of filter-based approaches is that there might be situations where all of the products are filtered out by the constraints [4], which is undesirable in practical settings. Therefore, ADVISOR SUITE implements a filter-relaxation algorithm based on the Hierarchical Constraint Satisfaction [17] technique. The system

evaluates the user inputs in the current state of the interaction and determines, which of the products should be proposed, i.e., which of the filter constraints have to be applied. When there are no products left or the number of remaining products does not reach a defined threshold, the system iteratively retracts filters until the desired number of products is reached. For that purpose, each of the filter constraints in the knowledge base is annotated with priority values which are typically defined by the domain expert; in many domains there are also strict rules that have to be obeyed, e.g., that one should not propose investments of high risk if the customer has no spare capital. Conversely, there might be non-strict expert rules, regarding, e.g., the major target industry sectors of an investment fund, which can potentially be ignored.

During the computation of the recommendation, the system keeps track both of the expert rules that are applied and those that were relaxed. When the result is finally presented to the customer, these rule sets are used in order to generate adequate explanations based on the natural-language annotations stored in the knowledge-base. Consequently, the explanations consist of a set of justifications like *"Given the information about your current financial situation, we recommend low-risk investments"* as well as explanations for relaxed rules like *"We also included products in the recommendation that do not match your wishes on the industry sectors of your investments."*

As the initial priorities defined by the domain experts do not necessarily match the preferences of customers, the users can interactively apply or relax filter constraints dynamically. If one of the applied expert rules is not important for a certain user, he can instruct the recommender engine to ignore it or force the application of a particular rule vice versa which gives the user the required degrees of freedom in the recommendation process.

Ranking. After the application of the filters a personalized ranking of the products according to the customer's

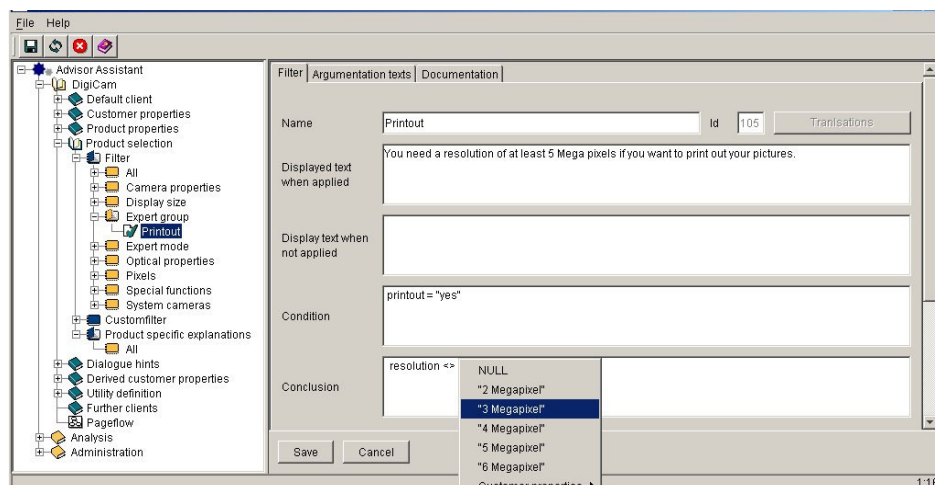


Figure 1: Graphical Knowledge Acquisition Tool

preferences is made. The initial ranking of the products is based on Multi-Attribute-Utility-Theory (MAUT) [23,1]. Similar to classical value-benefit analysis for product comparison, we can define several high-level product dimensions, like quality or economy, and define utility functions, i.e. relations from product properties to these ranking dimensions. While in classical value-benefit analysis the relative weight among the high-level dimensions is static or has to be entered manually, in the implemented MAUT approach, these relative weights are adjusted dynamically based on the customer's inputs. By using the questions about the user's preferences, the system derives the personal importance of the several product aspects for the customer and uses this data to subsequently personalize the product ranking.¹

Nonetheless, the design of ADVISOR SUITE offers the possibility to embed external algorithms for determining the ranking based on other customer's average rating, if such information is available.

3. Personalized Dialogues

Virtual (and real-world) sales assistance and product recommendation are highly interactive processes, typically dialogues consisting of questions and answers, hints and recommendations, whereby the dialogue flow must be adapted to the customer's expertise, and preferences. Depending on the customer's answers, the system has to pose different further questions and select a suitable interaction style with the customer. A typical example is the choice of technicality of the questions, depending e.g., on the user's self assessment of his expertise.

Therefore, in ADVISOR SUITE, the way how the system interacts with the user is regarded as another important piece of domain specific knowledge of a sales expert beside the core recommendation knowledge. Consequently, the knowledge-based approach is extended with a *conceptual model* for a declarative definition of web-based sales assistance dialogues:

- A recommendation dialogue consists of a set of *pages*; each one of them contains one or more questions, where the possible answers are presented in a given layout style, for instance as a radio button.
- The dialogue can optionally be organized in *phases*, in order to give the user an overview of the progress of the recommendation session.
- Within the application there exists a set of *special* pages, like result presentation, explanations, or additional hints.
- For each page we can define in a declarative way where to proceed, i.e., which page to display next,

whereby this decision depends on the user inputs. The evaluation of these conditions happens at run-time, when a *controller* page is invoked.

We intentionally used a conceptual model with a strong relation to the final web application to narrow the gap between the design model and the resulting application.

Dialogue design. Figure 2 depicts the graphical knowledge acquisition tool for defining the page flow of the sales assistance dialogue: On the left hand side, the designer of the application defines the individual phases and pages as well as the questions that have to be displayed on pages. In a detailed view, the designer can determine several parameters for each page, for instance, the style in which the question should be displayed.

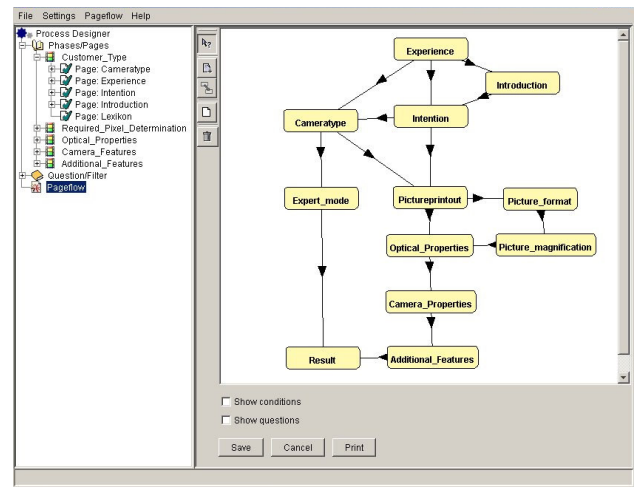


Figure 2: Modeling the interaction flow

On the right hand side, all possible paths through the interactive dialogue can be defined in a graphical way: vertices represent individual pages, edges link possible successor pages, whereby the links are annotated with transition conditions. These conditions are entered using the high-level language that is also used for the definition of filter constraints, e.g. "Follow this link, if the user stated to have low experience.". When using this style of page flows that depend on direct or indirectly derived inputs, it is possible to design a personalized interaction flow that immediately reacts on the user's behaviour and adapts the dialogue to the current situation. For modeling the flow of interaction, we do not directly rely on standard techniques for modeling dynamics in user interfaces, like State-Diagrams, Petri Nets, or UML-like extensions for modeling e-Commerce applications [15, 8], because our experiences have shown that domain experts and even web-site developers have significant problems in understanding these technical concepts. Instead, we decided to use a simplified, less technical notation similar to state diagrams, which ensures that this concept is understandable for domain experts, but still has a formal semantics.

¹ Further details on using MAUT for personalized user interaction can be found, e.g., in [1].

Hints. There are situations in real sales dialogues, where the sales advisor actively interrupts the dialogue. First, there might be conflicting answers, e.g., some questions of the dialogue may be used to cross-check the plausibility of previous answers. If such a conflict arises, the sales assistant will suggest his client to reconsider his answers. Second, another point of dialogue interruption can occur when the sales assistant offers additional hints or explanations; in real-life sales conversations these interruptions are also often used for cross- or up-selling purposes.

ADVISOR SUITE allows the designer of advisory applications to model interruptions of those two kinds, i.e., hints that relate to conflicting user input as well as hints that represent additional information for the customer. The moment in time when such a hint has to be displayed during the dialogue is modeled as a condition in the graphical knowledge acquisition tool, by using either the tool's standard constraint language or explicit tables of compatible and incompatible user input combinations

In general, the ADVISOR SUITE framework supports acquisition and maintenance arbitrary text fragments, e.g. for explanations. Each of these knowledge chunks can be maintained in different personalized variants, i.e., be annotated with a condition on the customer properties. At run-time, the correct personalized version of these informative texts is automatically selected by the system. Thus, maintenance of domain-specific texts can be carried out without working at the HTML-code level.

User interface generation. In order to accelerate the development process of the graphical user interface of an sales assistance system, a framework for automatic generation of dynamic HTML pages from the declarative definitions of the interaction flow was developed. This framework follows the Model-View-Controller² approach which strictly separates interaction control from presentation issues and the underlying repository content.

The web-based advisory application consists of generated *Java Server Pages*³ which are used for displaying questions, answers and informative texts and provide standard navigation which enables the customer to move freely through the dialogue. Additionally, the application includes a generic interaction module that handles user inputs (e.g. manages revisions of previously given answers), evaluates whether additional hints have to be given, and steers the interaction flow based on the definitions from the knowledge base. Furthermore, we make extensive use of *Custom Tags*⁴ such that the display of questions or possible answers can be performed in HTML like style without scripting code which in turn shortens the page adaptation process.

² <http://java.sun.com/blueprints/patterns/MVC.html>.

³ Java Server Pages, see <http://java.sun.com>.

⁴ Java Server Pages - Tag libraries, see <http://java.sun.com>.

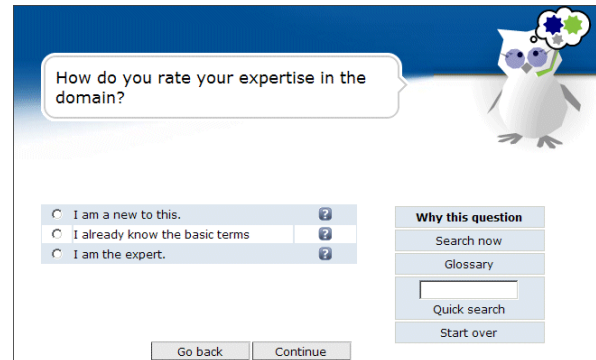


Figure 3: An automatically generated page

Figure 3 shows such a generated page that is built from predefined and easy-to-maintain templates. The problem of changes in the generated code is taken into account by the integration of an elaborated template mechanism for page generation. Additionally, hooks are integrated in the generated pages where custom code can be incorporated by the web developer, whereby this code will be unchanged in cases the pages are re-generated after maintenance activities, e.g., after a new page was introduced in the dialogue.

4. Architecture and Implementation

The overall system architecture is depicted in Figure 4. The complete knowledge for recommendation and personalization is maintained using graphical knowledge acquisition tools and stored in a central repository which is built on top of a relational database system. After generation of the user screens, the advisory application runs as application on a Web server, whereby for each customer an *Interaction Agent* manages the user interaction and performs the required personalization steps.

Integration with existing systems (like a Web-store or other enterprise applications) and extensibility are key issues for successful deployment of e-Commerce applications. Therefore, the system was built using Java-based technology as well as XML-based interfaces for data exchange. We did not rely on available expert system shells or special programming environments like *Prolog* in order to minimize the need for specialist developer knowledge. Our experiences show that this consistent use of state-of-the-art technologies significantly reduces the development and maintenance costs for expert systems with a web-based user interface. The usage of non-standard technology, for instance as a backend reasoner, would typically require programming experts and the implementation of additional interfaces between the knowledge-base, reasoner, and the user interaction components.

In the proposed system, performance issues commonly related with Java technology are addressed by ex-

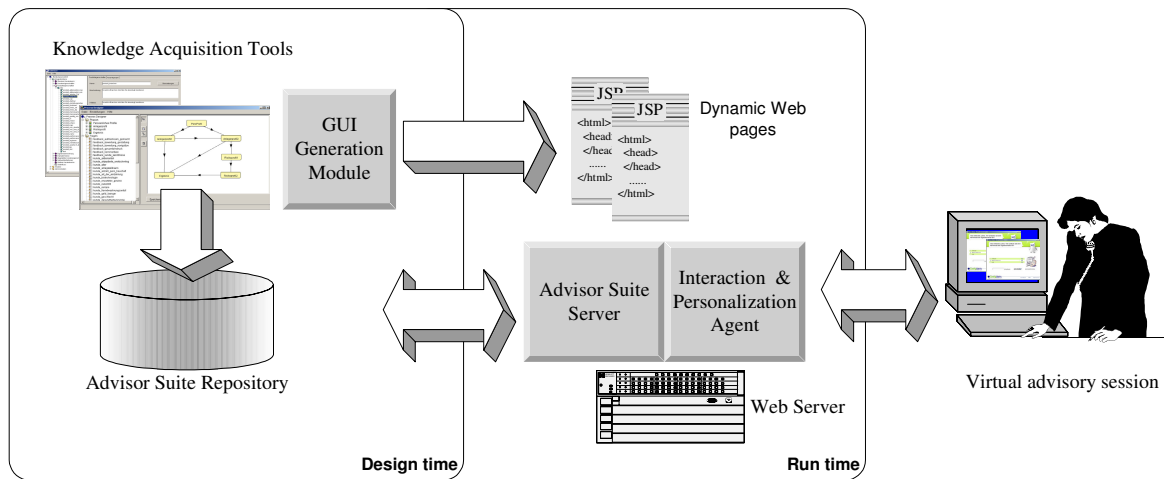


Figure 4 Overall architecture

tensive caching as well as pre-compilation of expert rules into an optimized internal representation.

5. Comparison

Recommendation technique. Over the last years, several techniques for personalized or non-personalized product recommendations have been successfully applied, whereby the most prominent examples are probably Amazon.com's⁵ or CDNow's⁶ online stores. From the perspective of the recommendation techniques, we can distinguish the following – often conjointly applied – approaches, compared to, e.g., [11, 5]:

- *Collaborative* These systems exploit explicit ratings on the available items made by the users. They also try to identify users that are similar to the current user and consequently extrapolate the items that are recommended.
- *Content-based* Compared to pure collaborative approaches, these systems utilize features of the items, like e.g., the genre of a book for user classification and for improving recommendation results.
- *Demographic* Beside the user ratings these systems also collect demographical and social information about the user to estimate the user's preferences [2].
- *Utility-based* They use information about item features and utility functions over the items that describe the user preferences.
- *Knowledge-based* These systems exploit explicitly acquired user needs and knowledge about feature items and about how these items match the user's preferences to infer recommendations.

All of these techniques have their specific advantages and problems. Collaborative techniques, for instance, are very well studied and have shown to be able to compute

recommendations that are appreciated by the users. Moreover, this technique does not require information about the products, i.e., no knowledge engineering or maintenance activities are required. However, a typical drawback of this technique is that no good recommendations can be made for new users or new items where no ratings exist. On the other hand, knowledge-based approaches make the expert knowledge explicit and allow the recommender system to generate plausible explanations of its recommendations, whereby for these systems possibly costly knowledge-engineering processes are required for acquiring both the expert and the product knowledge.

As a consequence, hybrid approaches are used to overcome the drawbacks of individual techniques [5]. In ADVISOR SUITE, such a hybrid approach was adopted. We exploit explicit expert knowledge for inferring user preferences by using direct and indirect questions and by filtering the available items based on a constraint-based technique. The remaining items are then ranked based on the expected utility for the user. In order to reduce the costs for knowledge elicitation, significant efforts have been made to simplify this process, e.g., by defining a high-level business rules language or by providing a convenient graphical user interface.

The main reasons for this choice lie in the targeted application domains, namely complex products or services. In these domains, e.g., electronic goods like digital cameras or investment decisions, the user typically would need deep domain knowledge about the items in order to select a product that matches his real preferences and demands. Even more, recommendations in these domains require good explanations for the suggested items to increase the customer's confidence in his buying decision.

User interaction. For a long time, research on recommender systems was mostly focused on the underlying algorithms that steer the product selection and recommendation processes. However, in particular when using content-based approaches that exploit knowledge

⁵ www.amazon.com

⁶ www.cdnw.com

about user preferences and product properties, more complex user interaction is needed. In turn, this makes dialogue design and dialogue efficiency important topics in the field [14, 19, 22, 20, 21, 9, 1].

Dialogue efficiency is directly associated to the dialogue's length, i.e., shortening the dialogue makes it more efficient and increases user satisfaction. Recent work on recommender systems based on case-based reasoning techniques for instance aim at improving the incremental requirements elicitation process in different ways. [14] describes an approach for a product attribute selection strategy that allows the system to terminate the dialogue prematurely without loss of solution quality. [19] proposes an adaptive selection strategy that can re-focus its recommendations based on user inputs dynamically.

Another aspect of recommender systems is personalization. Following the classification from [12], ADVISOR SUITE applications both adapt the content, the structure and the presentation of the sales dialogue. Beside the computation of personalized product proposals, the knowledge-based approach for adapting the dialogue flow according to the customer's answers allows us to personalize the communication by taking the customer's knowledge and preferences into account.

From our perspective, the final goal for online sales assistance dialogues is to provide the customer a natural language conversational user interface enhanced with a virtual character with which the user can establish an emotional relation. First ideas and approaches are for instance described in [21, 9] or [22]. In our view, the complexity of natural-language interfaces still hampers the application of such systems on a broad basis because of the high costs and set-up times. Therefore, we currently rely on adaptive, form-based user interaction with the possibility of including pages and hints which makes the dialogue more natural. Further work, however, will focus on incorporating mechanisms in this direction.

Application Design and Development. The ADVISOR SUITE framework enables the domain expert to completely design the recommendation dialogue using graphical tools and to automatically generate a web-based application. This allows us to bridge the gap between requirements elicitation and the subsequent development phases. Existing model-based approaches like, for instance WebML or OO-HMethod [6, 7, 10, 15], aim at providing general methodologies for designing web applications on a conceptual level. Compared to that work, ADVISOR SUITE is limited to web-based sales advisory applications. This allows us to use a specific, simplified modelling notation which also allows domain experts to specify the dialogue flow. In addition, the designer only has to model one single dialogue flow (steered by the users' inputs) instead of multiple flows for different contexts, e.g. for different users or user groups [7], which also improves simplicity and clarity. In contrast to general conceptual

modeling approaches, the automatic generation of the recommender application in our system allows us to immediately test and use of the application, even during the prototyping phase, whereas purely conceptual approaches potentially exhibit a gap between the design model and the final web application.

6. Experiences from Practical Settings

Up to now, several instances of advisory applications built with ADVISOR SUITE system were deployed in various domains like in the financial sector, for "technical" goods like digital cameras or skis, as well as for "quality-and-taste" domains like cigars. From the development perspective we encountered that the development times for the core application are in fact very short and the basic knowledge-base could be developed in a few workshop days. Note that the initial knowledge bases were typically not created by the domain expert alone, but together with a knowledge engineer. After this phase, however, the expert was able to carry out the necessary maintenance tasks on his own requiring the engineer's help only in a few cases. Quite interestingly, the number of business rules defined by the experts is rather small, i.e., only a few dozen rules, which is a promising small number with respect to overall knowledge acquisition and maintenance costs. In this context, our experiences have shown that the modeling language and the graphical notation are easy to comprehend for the domain expert. Additionally, the automatic generation of the application is very helpful especially in the set-up phase of the knowledgebase, where the impact of changes can be tested immediately. Furthermore, the structure and the layout of the templates and the generated pages have shown to be simple enough to be adapted by a web developer in order to fit the corporate design of a company.

One of the key factors of the acceptance of advisory applications by end users lies in the quality, up-to-dateness, and completeness of the underlying product data. In most cases, a major part of the data is already available in electronic form but had to be manually enriched, e.g., with additional properties. Both the real-time access of product data in existing databases and the periodical data import via the provided XML interface ensure a satisfying quality of the data. Regarding user acceptance, we measured that in a project on one of Austria's largest e-Commerce sites with respect to daily users, over eighty percent of the sessions were successful, i.e., the customers stepped through the whole dialogue to the result page showing the proposals.

As a side effect, companies offering online advisory systems do not only profit from increased customer satisfaction by the value adding online service, but also from the new information provided by the system. Since all information about user interactions are stored and can be

evaluated online, they can learn about their customers in the sense of improved Customer Relationship Management. For instance, knowing whether one's online customers rate themselves to be experts, can serve as an aid for the company to tailor the online service to this target group. On the other hand, the evaluation of click-behaviour of the online users can help us improving the advisory application itself, e.g., we can determine when the dialogue is terminated by the user prematurely. Future work will therefore include advanced web mining techniques for automatic identification of such situations.

7. Conclusion

Product recommendation and virtual sales assistance are areas that can create significant customer benefit and improve customer relations on the online channel. We have presented a software framework for rapid development of personalized, interactive advisory systems for arbitrary domains following a content- and knowledge-based approach. Our practical experiences have shown that a knowledge-intensive approach can be successful, if adequate graphical knowledge acquisition and maintenance tools are available and the underlying concepts are presented to the domain expert using an easy-to-understand terminology. Finally the usage of a common development technology throughout the system allows us to apply standard industrial software engineering practices for the development of an expert system that has to be tightly integrated with a web-based user interface.

References

- [1] L. Ardissono, A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, and R. Schäfer. A framework for the development of personalized, distributed web-based configuration systems. *AI Magazine*, 24(3):97-110, 2003.
- [2] C. Basu, H. Hirsh, and W. W. Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *AAAI/IAAI*, pp. 714-720, 1998.
- [3] R. Bergmann, R. Traphoner, R. Schmitt, P. Cunningham, B., and Smyth. Knowledge-intensive product search and customization in electronic commerce. In *E-Business Applications*. Springer Verlag, 2003.
- [4] D. Bridge. Product recommendation systems: A new direction. In R. Weber and C. Wangenheim, editors, *Procs. of the Workshop Programme at the Fourth International Conference on Case-Based Reasoning*, pp. 79-86, 2001.
- [5] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, (4):331-370, 2002.
- [6] S. Ceri, P. Fraternali, and A. Bongio. *Web Modeling Language (WebML): a modelling language for designing Web sites*. *Computer Networks* 33:137-157, 2000.
- [7] J. Gómez, C. Cachero and O. Pastor. Extending a Conceptual Modeling Approach to Web Application Design. In *Proceedings of the 1st International Workshop on Web-Oriented Software Technology*, Valencia, Spain, June 2001.
- [8] M. Green. A Survey of Three Dialogue Models. In: *ACM Transactions on Graphics*, 5(3):244-275, 1986.
- [9] T. Gurzki, P. Schweizer, and C.-T. Eberhardt. A virtual-sales-assistant architecture for e-business environments. In *E-Business Applications*, pp. 77-86. Springer Verlag, 2003.
- [10] M. D. Jacyntho, D. Schwabe, and G. Rossi. A Software Architecture for Structuring Complex Web Applications. *Journal of Web Engineering* 1(1):37-60, 2002.
- [11] A. Jameson, J. Konstan, and J. Riedl. AI Techniques for Personalized Recommendation. Tutorial Notes. In *18th International Conference on Artificial Intelligence*, Acapulco, Mexico, August 2003.
- [12] A. Kobsa, J. Koenemann, and W. Pohl. Personalized hypermedia presentation techniques for improving online customer relationships. *The Knowledge Engineering Review*, 16(2):11-155, 2001.
- [13] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77-87, 1997.
- [14] D. McSherry. Increasing dialogue efficiency in case-based reasoning without loss of solution quality. In *18th International Joint Conference on Artificial Intelligence*, pp. 121-126, Acapulco, Mexico, August 2003. Morgan Kaufman.
- [15] O. Pastor, S. Abrahao, and J. Fons. Building e-commerce applications from object-oriented conceptual models. *SI-Gecom Exchanges*, 4(2):28-36, 2001.
- [16] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56-58, 1997.
- [17] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *International Joint Conference on Artificial Intelligence*, pp. 631-639, Montreal, Canada, 1995.
- [18] J. J. B. Shafer, J. Konstan. Recommender systems in e-commerce. In *ACM Conference on Electronic Commerce (EC-99)*, pp. 158-166, 2001.
- [19] B. Smyth and L. McGinty. The power of suggestion. In *18th International Joint Conference on Artificial Intelligence*, pp. 127-132, Acapulco, Mexico, August 2003. Morgan Kaufman.
- [20] K. Swearingen and R. Sinha. Beyond algorithms: An HCI perspective on recommender systems. In *ACM SIGIR 2001 Workshop on Recommender Systems*, New Orleans, Louisiana, 2001.
- [21] U. Thiel, M. Abbate, A. Paradiso, A. Stein, G. Semeraro, and F. Abbattista. Intelligent e-commerce with guiding agents based on personalized interaction tools. In *E-Business Applications*, pages 61-76. Springer Verlag, 2003.
- [22] C. A. Thompson, M. Goeker, and P. Langley. A personalized system for conversational recommendations. In *Technical Report UUCS-02-013*, School of Computer Science, University of Utah, Salt Lake City, 2002.
- [23] D. von Winterfeldt and W. Edwards. *Decision Analysis and Behavioral Research*. Cambridge University Press, Cambridge, UK, 1986.

Case Study Methodology Designed Research in Software Engineering Methodology Validation

Seok Won Lee¹ and David C. Rine^{2,3}

¹*Dept. of Software and Information Systems, University of North Carolina at Charlotte
9201 University City Blvd, Charlotte, NC 28223, USA.*

²*Dept. of Computer Science, George Mason University, Fairfax, VA 22030, USA.*

³*NASA NET Research Group, George Mason University, USA.
seoklee@uncc.edu, drine@gmu.edu*

Abstract. One of the challenging research problems in validating a software engineering methodology (SEM), and a part of its validation process, is to answer “How to fairly collect, present and analyze the data?”. This problem adds complexity, in general, when the SEM involves the use of human knowledge in its methods (phases). How should such created knowledge be captured in the methodology during a SEM process? How can such knowledge be made available for continued SEM process improvement? How can such knowledge be used in validating the SEM? Measuring such knowledge is hard, but we can benefit from the “Case study research design” which is a valuable and an important empirical research alternative in designing a research plan that establishes a logical link from the data to be collected to the initial questions of study. In this paper, a case study research methodology (CSM) designed is presented with its application to the validation of a software requirements engineering methodology (SREM). The preliminary results show the evidence used to validate the SREM as well as the potential usage of CSM as a goal-oriented research design, practice and teaching methodology.

1. Introduction

A case study design, as a technology empirical evaluation research methodology and a way to generalize from observed case study outcomes, builds a basis for valid inferences from the case study events and evidence collected. An invented SEM may be such a technology. For an effective research case study, as an empirical research methodology applied as a validation exercise – applied to an ‘invented software (systems) engineering methodology,’ it is necessary for the validation exercise to first have designed a case study methodology specific to the characteristics of this invented SEM. More specifically, those characteristics are: 1) the characteristics of this invented SEM that required interventions from the domain Subject Matter Expert (SMEs) or the software engineer in order to perform on demand each appropriate step in the SEM; and 2) the characteristics of the invented SEM validation that cannot

favor the invented SEM over alternatives because of the uniqueness of the invented methodology or the relative different level of understanding of the domain and analytical skill of SMEs in the actual case study ‘experimental’ conditions. Therefore, the consideration of these two above characteristics motivated development of the case study design in this paper, based on the theories and guidelines from [11]. In addition, the theoretical framework of the case study and the application of this case study to the invented Proxy Viewpoints Model-based Requirements Discovery (PVRD) methodology [5, 6] will provide better understanding of the PVRD methodology, and guidance to researchers from academia and real practitioners from industry.

In this paper, the invented SEM (technology) to be validated by a case study empirical approach is the PVRD methodology. In the following sections, a brief introduction of the PVRD methodology and the components in the research case study design will be outlined.

2. The PVRD Methodology

The Proxy Viewpoints Model-based Requirements Discovery (PVRD) is a methodology [5, 6] that provides an integrated framework to reason about “missing natural language system requirements” problems. The PVRD methodology consists of four models: viewpoints model, enterprise model, missing requirements types categorization model, and requirements discovery and analysis model. The viewpoints model [7, 4, 9] represents different perspectives or views for a coverage of direct and indirect stakeholders that need to be identified and incorporated into the legacy status software system requirements. The enterprise model [1] provides a way of categorizing requirements based on systems engineering design process models. The missing requirements types categorization model provides a method to project a requirements space that may contain specific types of missing requirements. The requirements discovery and analysis model provides a method to retrieve requirements of interest by using the requirements term

expansion method [8, 2] that automatically generates a list of “potential query terms” [10] which could assist analysts in acquiring more knowledge about the domain of interest by performing a “complete search” of available requirements resources.

Based on this integrated framework, the PVRD methodology is able to create a proxy viewpoints model and provide a new way of discovering missing natural language system requirements while improving the legacy natural language requirements representation space through the modeling of a new indexing structure that supports multiple viewpoints from many stakeholders in a large-scale complex software system.

The PVRD methodology is applicable in the use of existing natural language software requirements specifications (SRS) in further improved development of legacy systems by 1) discovering missing requirements, especially, when it is necessary to reconstruct the original legacy SRS, and 2) eliciting new requirements for system changes that will take place or creating a new system from a similar legacy system. Figure 1 shows the overview of the steps in the PVRD methodology. More detailed descriptions of the PVRD methodology are in [5, 6].

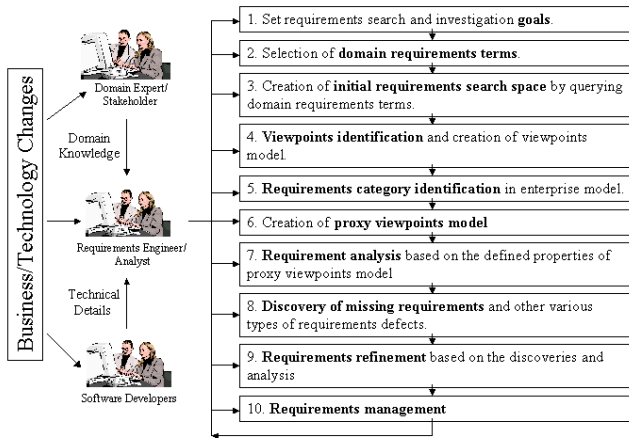


Figure 1. Overview of the PVRD Methodology Steps

3. Case Study Research Design Components

The goal of the PVRD research is “the development of a new methodology that can discover missing natural language requirements and reduce the number of incomplete requirements while reorganizing the requirements representation space that incorporates and supports multiple viewpoints in legacy status, large-scale, information system requirements specifications”.

In order to show *how* and *why* (“explanatory” type of case study in [11]) the PVRD methodology can achieve this research goal, the following five important

components will be defined during the case study design process [11]:

- A study’s questions,
- Study propositions,
- Unit(s) of analysis,
- The logic linking of the data to the propositions, and
- The criteria for interpreting the findings.

Figure 2 shows the inter-relationships between these five components in the explanatory case study design. The study questions can be mapped and further decomposed into a set of more detailed study propositions. These propositions contain metric terms and are used to develop measure data capture questionnaires. The application by the SMEs of the developed PVRD methodology to the units of analysis can then generate results observed and reported by the SMEs in the questionnaires, i.e. measure data collection instruments. The results are then linked back to the study propositions as evidence through the criteria (using metrics) for interpreting these findings.

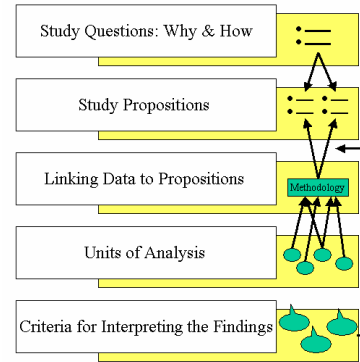


Figure 2. Case Study Design Components

3.1. Study Question

Study questions need to be clarified precisely [11]. For instance, in this case study design, the study question “How and why can the PVRD methodology discover missing requirements from natural language software requirements specification documents that involve multiple viewpoints while reorganizing and improving the quality of a requirements representation space?” needs to be clarified by further decomposition of that research question into sub questions or propositions. In this case study, these natural language software requirements specification documents identified in this study question are the case study units of analysis.

3.2. Case Study Propositions

Case study propositions, derived from the case study questions, become assertions that should be examined, through measure-valued questionnaire items, to answer a

study question within the scope of the case study. For instance, in this case study design, the *general study proposition* is “the PVRD methodology can achieve its research goals because of its integrated framework that benefits by the synergism between the embedded models and methods”. And four more *specific case study propositions* derived from this proposition are as follows:

The PVRD methodology and its integrated framework can 1) Reduce the number of incomplete requirements by discovering missing requirements; 2) Discover requirements defects of various types; 3) Discover requirements relationship and workflow process relationship chains in the requirements space; and 4) Create new requirements indexing structures based on the embedded models.

3.3. Units of Analysis

Units of analysis are the selected resources to be examined through the application of the invented technology (e.g. PVRD methodology) by the SMEs in the case study. For instance, in the case study methodology designed for the research reported here, the PVRD methodology will be applied to the (units) “set of software requirements specification documents” that is expressed in natural language. These requirements (units of analysis) are from the legacy status information-based software system that includes many stakeholders (e.g. interactive systems). Also the SMEs and other software development resources (other than SRSs, such as business process descriptions, operational concepts etc.) need to be involved due to their influences in the requirements classification (e.g. assigning requirements to viewpoints) of each model (i.e. Viewpoints Model, Enterprise Model, Missing Requirements Types Categorization Model, Requirements Discovery and Analysis Model) in the PVRD methodology. From the PVRD methodology point of view, the models and methods in the PVRD methodology, and the properties associated with the PVRD methodology, are the subjects to be examined during a case study using this case study design.

Table 1 presents the ‘Requirements Discovery Summary Sheet’ (RDSS) that is used by a team of SMEs during a case study using this case study design. Each step in the RDSS sheet corresponds to the steps in Figure 1. This RDSS sheet is utilized for SMEs to serve the following roles during a case study:

- Provides a template of how to apply the PVRD methodology to the units of analysis in a case study. Combined with the given instructions during a case study, SMEs can identify the units that need to be examined (analyzed) and results to be recorded.
- Captures evidence and findings using the criteria defined by metrics with measures (combination of qualitative and quantitative analysis in section 3.5)

that will be used to support/reject the case study propositions (in section 3.2). Without having such specific propositions with metric criteria word attributes in them, an investigator might be tempted to collect “everything” which is impossible to collect [11] and meaningless. For instance, as shown in Table 1, the results from the ‘Discovery types’ [SID-8-DID-1] and ‘Level of significance of discovery’ [SID-8-DID-2] from the RDSS sheet will be used to support/reject the case study propositions 1, 2, and 3. Also the results from the ‘Viewpoints identification’ [SID-4-1], ‘Requirements category identification in enterprise model’ [SID-5-1], ‘Proxy viewpoints model creation’ [SID-6-1], and ‘Newly indexed requirements’ [SID-8-DID-7] will be used to support/reject the case study proposition 4.

- Guides SMEs in a step-by-step approach while conducting the PVRD methodology case study. SMEs follow each step of the PVRD methodology and record their findings and observations (unit by unit) under each step in the RDSS sheet.

In Table 1, steps 1 – 6 focus on the process of the proxy viewpoints model creation, and steps 7 – 8 focus on the requirements analysis and discovery process based on the created proxy viewpoints model. Therefore, the questions under each step from 1 to 6 and the evidence collected by SMEs would capture the idea of “how SMEs created the proxy viewpoints model from the given requirements set”. In step 7, SMEs collect their units PVRD methodology analysis results from the created proxy viewpoints model. The fine-grained questions (units) in step 8 would capture the specific evidence to support/reject the propositions related to the requirements discovery process. In the questionnaire, some questions (i.e. SID-8-DID-2, SID-8-DID-7, SID-8-DID-8) will be interpreted based on the qualitative measures and some questions (i.e. SID-8-DID-1) will be interpreted based on the quantitative measures. The following Table 1 summarizes important aspects of the RDSS sheet.

In addition to collecting specific evidence to support/reject specific propositions, it is also important to collect evidence of the ‘entire process’ wherein the methodology is applied to the given requirements set (units of analysis). This is because the specific evidence is captured in the middle or after the application of the PVRD methodology, and it did not come from an independent evidence collection process. Also the collection of evidence from the ‘entire process’ must be used, wherein the PVRD methodology is used to support/reject the general proposition in section 3.2 (also the study question in section 3.1).

Having the RDSS with clearly identified steps and interpretation criteria (metric and measures) is important and related to the general ‘repeatability’ of the case study methodology.

Table 1. Evidence Collection through Requirements Discovery Summary Sheet

Questions (Units)	Evidence captured by SMEs (Requirements Discovery Summary Sheet - RDSS)	Step/Model/Method in the PVRD	Related Propositions that support/reject
SID-1-1	SMEs record the <u>goal(s)</u> of requirements search/investigation.	Step 1	General Proposition (GP)
SID-2-1 SID-2-2 SID-2-3	SMEs record the selected ' <u>key domain terms</u> '. SMEs record the <u>reason</u> for the selected terms. SMEs record the specific type of ' <u>missing requirements types</u> ' if it is used in the selection of domain requirements terms SID-2-1 (with explanation).	Step 2, Missing Requirements Types Categorization Model	GP
SID-3-1	SMEs record the number of requirements in the <u>initial requirements search space</u> created (with requirement ID).	Step 3	GP
SID-4-1	SMEs record identified <u>viewpoints (VP)</u> for each requirement with its ID.	Step 4, Viewpoints Model	GP, Proposition 4
SID-5-1	SMEs record identified <u>category of enterprise model (EM)</u> for each requirement with its ID.	Step 5, Enterprise Model	GP, Proposition 4
SID-6-1	SMEs check each requirement index based on the VP and EM and create a <u>proxy viewpoints model</u> and the layout.	Step 6, Proxy Viewpoints Model	GP, Proposition 4
SID-7-1	SMEs analyze the PVRD layout and record any <u>discovery patterns</u> found in the created proxy viewpoints model.	Step 7, Requirements Discovery and Analysis Model	GP
SID-8-DID-1	SMEs record the <u>types of discovery patterns</u> found in the created proxy viewpoints model.	Step 8, Requirements Discovery and Analysis Model	GP, Propositions 1, 2 and 3
SID-8-DID-2	SMEs record the <u>level of significance</u> of the discovery patterns found.	Step 8, Requirements Discovery and Analysis Model	GP, Propositions 1, 2 and 3
SID-8-DID-3	If the <u>term expansion method</u> is used in the discovery process, SMEs record specific steps taken (with detailed explanation of how this method is used and contributed to this discovery process).	Step 8, Term Expansion Method, Requirements Discovery and Analysis Model	GP
SID-8-DID-4	SMEs record the ' <u>requirements distance</u> ' from this discovery.	Step 8, Requirements Discovery and Analysis Model	GP
SID-8-DID-5	If ' <u>missing requirements types</u> ' are used in this discovery, SMEs record the specific type and explanation of how it is used in this discovery.	Step 8, Missing Requirements Types Categorization Model, Requirements Discovery and Analysis Model	GP
SID-8-DID-6	If <u>new types of missing requirements</u> are discovered, SMEs record this new type and specific explanation of what they are.	Step 8, Missing Requirements Types Categorization Model, Requirements Discovery and Analysis Model	GP
SID-8-DID-7	SMEs record their observation about the <u>newly indexed requirements representation</u> (through the VP and EM), compared to the original requirements structure.	Step 8, Proxy Viewpoints Model	GP, Proposition 4
SID-8-DID-8	SMEs record their observation about the <u>PVRD methodology contribution for this discovery</u> .	Step 8, Proxy Viewpoints Model, Requirements Discovery and Analysis Model	GP
SID-8-DID-9	SMEs record their comments about their <u>experience with the PVRD methodology for this discovery</u> .	Step 8, Proxy Viewpoints Model, Requirements Discovery and Analysis Model	GP

3.4. Linking Data to Study Propositions

Linking data to propositions represents the data analysis step in the case study design research [11]. The PVRD methodology is applied to the units of analysis and plays a role in connecting the generated measure data results back to the study propositions.

In a case study using this case study design, the generated measure data results can be based on any mix of *qualitative* and *quantitative* evidence. In this case study, the generated results will be collected by SMEs in the form of RDSS sheet in Table 1 and also the notes from the lessons learned meeting. In addition, a case study need not always include a direct, detailed observation as a source of evidence [11]. Therefore, in

this case study, a set of evidence, its analytical interpretations, and lessons learned are sources of evidence as summarized in Table 1.

From the findings through the RDSS sheet, Table 1 serves as qualitative and quantitative evidence, such as 1) whether or not the discovered missing requirements are defining, mandatory or optional requirements (as qualitative measures defined in section 3.5); and 2) numbers and types of discoveries found (as quantitative measures).

Figure 3 shows how the findings are linked in the support of corresponding study propositions (an explosion of the link between the 'study propositions' and 'linking data to propositions' in Figure 2). The findings will be collected through the RDSS sheet in Table 1 by SMEs during a case study.

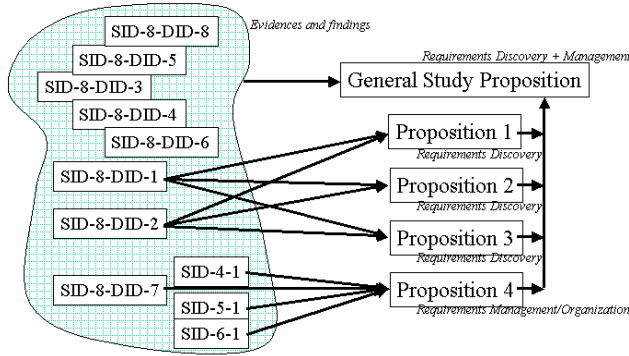


Figure 3. Linking Data to Study Propositions through Discovery Summary

3.5. Criteria for Interpreting a Case Study's Findings

Criteria for interpreting a case study's findings correspond to the metric and measures used in evaluating the results from the properties of requirements defects types defined in the PVRD methodology (such as incomplete, inconsistent, redundant, and ambiguous, as well as requirements relationship chain and workflow process relationship chain). In other words, the results from the requirements discovery and analysis model [5, 6] focus on the findings of significant defects of requirements and improvement of requirements quality as well as the quantitative analysis of how many requirements defects discovered. One example of such findings of significant defects that will be focused on is whether or not the discovered missing requirements or defects are one of defining, mandatory or optional requirements. The discovery of defining or mandatory requirements is much more critical than discovery of optional requirements. Therefore, the "importance/significance of the discovered requirements defects" serves as a *metric* in the analysis of the findings and three different requirements types "defining, mandatory or optional requirements" serve as *qualitative measures* in deciding the significance of the requirements. Also the number of requirements defects of various types serves as *quantitative measures*. Therefore, the metrics and measures will be all interpreted from a combined qualitative and quantitative analysis perspectives based on the summary of RDSS sheet in Table 1. All five components in the case study design described will guide a case study and become the fundamental basis in validating the PVRD methodology.

4. Multiple Case Studies

One of the most important points made in Yin's case study design approach [11] is the design of a theoretical case study framework which is presented in section 3.

Also, it is important to understand the importance of 'analytical generalization' – case studies (as with experiments), compared to 'statistical generalization' – survey research, in case study design. In statistical generalization, an inference is made about a population (or universe) on the basis of empirical data collected about a sample (i.e. surveys). In analytical generalization, the investigator is striving to generalize a particular set of results to some broader theory [11].

The evidence from multiple cases is often considered more compelling, and the overall study is therefore regarded as being more robust [3, 11]. A theory must be tested through replications of the findings in a second or more case that will lead to an analytical generalization. Under the development of a theoretical framework in [11], a *literal replication* (each case predicts the similar results) can explain the conditions under which a particular phenomenon is likely to be found, a *theoretical replication* (each case produces contrasting results but for predictable reasons) can explain the conditions when it is not likely to be found. Multiple case studies were carried out for the PVRD methodology validation and established a literal replication. For each individual case, collected evidence indicated how and why a particular proposition was demonstrated (or not demonstrated).

5. A Case Study in Educational Information Management System (EMS)

This particular case study is performed by a team of domain independent SMEs from industry (more than 10 years software requirements engineering & software development experiences) based on the CSM in order to formally confirm and validate the case study propositions of the PVRD methodology. For this case study, SMEs are trained to understand and apply the PVRD methodology through an orientation/workshop to practice performing a case study independently. The purposes of the workshop/orientation are to educate SMEs about the PVRD methodology case study, the methodology, embedded models, and methods, and going through step-by-step approach. It is also important that the researcher conducting the case study must have no interaction with the SMEs once the case study exercise is underway so as not to bias or prejudice SMEs' judgement. The EMS requirements documents size was over 300 pages. The findings and evidence are recorded and collected through the RDSS sheet using a given set of instructions.

Figure 4 shows the PVRD model that SMEs constructed during this case study in which became the basis of the discovery process. Each requirement represented in this PVRD model has been applied to each model and method embedded in the methodology, and corresponding relationships are established across the models.

By using the designed case study methodology as presented in this paper, evidence that can support/reject the case study propositions is collected as well as all the detailed observational descriptions by SMEs. Three investigations' results are summarized in Table 2. Other case studies that were carried out (but are not reported in this paper), more detailed description of the case and experimental set up are in [5].

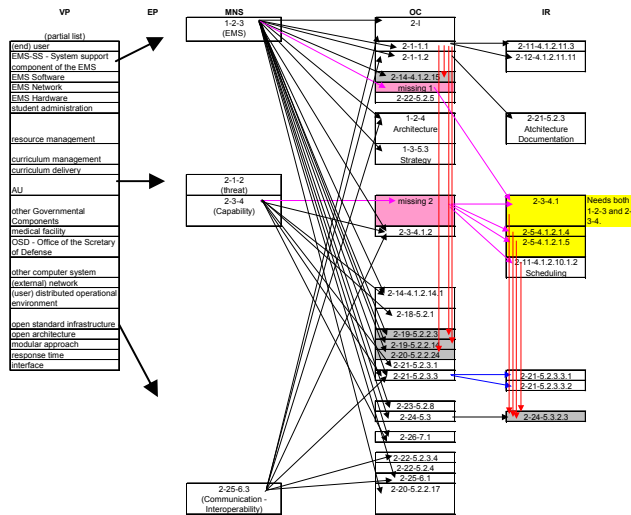


Figure 4. The PVRD Model

Table 2. EMS Case Study Results Summary

Results Category	Investigation 1 (SME #1)	Investigation 2 (SME #1)	Investigation 3 (SME #2)
# Discoveries	4	3	2
Missing Reqts	1	1	1
Other types of defects	Redundant (1) Inconsistent/ Ambiguous (1)	Redundant (1) Inconsistent/ Ambiguous (1)	
Reqts Relationship Chains	1 (Requirements relationship)		1 (Workflow process relationship)
# 'Defining' Reqts Discovery	1		
# 'Mandatory' Reqts Discovery	2	1	2
# 'Optional' Reqts Discovery	1	2	
New missing reqts types category, reqts distance, etc.	New missing reqts type, Requirements distance (>20)	Requirements distance (>20)	Requirements distance (>20)
The PVRD methodology contribution to the discovery process	Strongly (1) – missing requirements, Moderately (3) – others	Strongly (1) – missing requirements, Moderately (2) – others	Strongly (2)
Supporting Propositions	General, Propositions 1,2,3, and 4	General, Propositions 1,2, and 4	General, Propositions 1, 3, and 4
Rejecting Propositions		Proposition 3	Proposition 2
Overall Supporting Propositions	General Proposition, Propositions 1,2,3 and 4		

6. Conclusion and Future Work

Case studies are multi-perspective/dimensional analyses that need to consider many aspects in collecting evidences from various resources during their design and executions. For example, the use of the CSM in validation of the PVRD methodology considers not only the technical aspects but also the interactions with SMEs in capturing data and knowledge acquired.

The CSM can leverage its usage as follows: 1) a “goal-oriented” research design, practice and validation methodology (through effective evidence collection, presentation and analysis); 2) a flexible but theoretically powerful and solid methodology that can cover various interdisciplinary research domains' characteristics; and 3) a teaching methodology in education to cultivate student's integrative analytical and problem-solving skills. As a future work, more in-depth study of the application of the CSM to knowledge-intensive software engineering methodology validation is planned.

References

- [1] Buede, D. M. *The Engineering Design of Systems : Models and Methods*, New York: Wiley, December. 1999.
- [2] Faloutsos, C. and Oard, D., A Survey of Information Retrieval and Filtering Methods, CS-TR-3514, University of Maryland, 1995.
- [3] Herriott, R.E. and Firestone, W.A. Multisite qualitative policy research: Optimizing description and generalizability. *Educational Researcher*, 12, 14-19. 1983.
- [4] Kotonya, G. and Sommerville, I. Requirements Engineering with Viewpoints, *BCS/IEEE Software Engineering Journal*, Vol. 11, No. 1, pp.5-18. 1996.
- [5] Lee, S. W. “Proxy Viewpoints Model-based Requirements Discovery” *PhD Dissertation*, George Mason University, Fairfax, VA. 2003.
- [6] Lee, S.W. and Rine, D.C. “Missing Requirements and Relationship Discovery through Proxy Viewpoints Model,” In *Proceedings of the 19th annual ACM Symposium on Applied Computing (SAC 2004)*, Software Engineering, March, Cyprus, ACM Press, 2004.
- [7] Nuseibeh, B., Kramer, J. and Finkelstein, A. A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification, *IEEE Trans on Software Eng*, 20(10):760-773, IEEE CS Press, October. 1994.
- [8] Salton, G. *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice-Hall Inc., Englewood Cliffs, New Jersey. 1971.
- [9] Sommerville, I. and Sawyer, P. Viewpoints: Principles, Problems and a Practical Approach to Requirements Engineering. *Annual Soft Eng*, Vol. 3. pp. 101-130. 1997.
- [10] Xu, J. and Croft, B. Improving the Effectiveness of Information Retrieval with Local Context Analysis. *ACM Trans on Info Systems*, Vol.18, No.1, pp.79-112. Jan. 2000.
- [11] Yin, Robert. K. 1994. *Case Study Research: Design and Methods*. 2nd Edition. Thousand Oaks: Sage Publications, Applied Social Research Methods Series Vol. 5.

Data-mining in Support of Detecting Class Co-evolution

Zhenchang Xing and Eleni Stroulia
Computing Science Department
University of Alberta
Edmonton AB, T6G 2H1, Canada
{xing, stroulia}@cs.ualberta.ca

Abstract

In an evolving system maintained over a long time period, there exist many non-trivial relationships among system classes, such as class co-evolutions, which usually are not easily perceivable in the source code. However, unfortunately, the continuing evolution of large, long-lived systems leads to lost information about these hidden relationships. In this paper, we propose a method for recovering such lost knowledge by data mining method. This method relies on the UMLDiff algorithm that, given a sequence of UML class models of a system, surfaces the design-level changes over its life span, thus eliminating the need for high quality modification reports and non-intuitive software code-based metrics. We employ Apriori association rule mining algorithm to the transactional database of class modifications, which elicit previously unknown or undocumented co-evolving relations among two or more classes. The recovered knowledge facilitates the overall understanding of system evolution and the planning of future maintaining activities. We report on one real world case study evaluating our approach.

1 Introduction

The objective of software reverse engineering is most often to gain a sufficient design-level understanding in support of maintenance, adaptation and feature extension [6]. In object-oriented systems, classes model abstractions of real-world entities around which these systems are designed. In an evolving system maintained over a long period, there exist many non-trivial relationships among system classes, which may not be intentional and usually are not easily perceivable in the source code.

A particularly interesting such relation is class co-evolution. Frequently, classes not explicitly related in the system design exhibit parallel evolution history. This phenomenon may indicate an implicit inter-dependence which, when understood, can be valuable in guiding subsequent evolution of the system in question. First, the system maintainers may decide to restructure the system in order to eliminate this interdependence, thus evolving it into a more modular and less coupled design. Alternatively, they may document the inter-dependence as a predictor of maintenance activities, so that when some of the co-

evolving classes have to be modified the rest of the cluster is also examined and retested.

Recovering and making explicit such “lost knowledge” to increase the overall comprehensibility of a given system is one of the major objectives of reverse-engineering research. And, as many have already recognized [3,13], this task can benefit from Artificial Intelligence (AI), and more specifically data-mining, techniques. More specifically, Shirabad [15] recently proposed a method based on inductive-learning algorithm to address the problem of detecting the co-evolution of two code modules. Based on past maintenance experience, recorded in the form of change requests and code-update records, he explored the supervised inductive-learning method for recognizing co-updated modules and using this relation to predict whether updating one source file may require a change in another file.

An important shortcoming of this work is its knowledge requirements. It essentially assumes the existence of a fairly detailed change-tracking system in which all change requests are recorded. Then these requests are co-related with the code updates committed in response to “closing” the requests. These co-related requests and updates become the examples input to the learning algorithms. Unfortunately, however, such consistently kept change-tracking systems are not always available [6].

In our work on detecting class co-evolution, we have adopted class-design models of subsequent system snapshots (which may be released versions or simply snapshots checked-out in regular time intervals) as the primary input of our method. These class-design models are easily obtainable, given a version-management system and any of a variety of existing round-trip software-development tools [23,24]. The fundamental intuition underlying our class co-evolution detection method is that by comparing a sequence of snapshots of system’s class models, one can extract a history of the evolution of each individual class in terms of the “additions”, “deletions”, “moves”, “renamings” and “signature changes” of its super- and sub- classes, interfaces, and their fields and methods. Then rule- or sequence- mining algorithms, such as Apriori for example, can be used to detect common change co-occurrences among these class histories, thus uncovering co-evolving classes.

In addition to its simpler knowledge assumptions, our approach exhibits two advantages over Shirabad’s method. First it is unsupervised: unlike Shirabad’s method that requires a set of co-evolution examples in the form of sets of modules that were updated for the same change request, our method does not require labeled training examples. Second, it is relatively more scalable because it focuses on the changing system classes instead of all its modules.

The remainder of the paper is structured as follows. Section 2 relates this work to previous researches. Section 3 presents the overall methodology and rationale of our approach. One case study illustrating our approach is discussed in section 4. Finally, Section 5 concludes with a summary of the lessons we have learned to date and our plans for future work.

2 Related work

Our class co-evolution detection work spans over two related-research themes: first, the general area of employing artificial-intelligence methods in support of software reverse engineering, and second, the semantic manipulation of UML design models.

2.1 AI in support of reverse engineering

Artificial-intelligence methods can benefit several reverse-engineering processes, and design recovery in particular. We have already discussed the work most similar to ours in terms of objectives and types of AI algorithms employed.

Shirabad et al. [15] applied inductive machine learning method to elicit co-update file pairs of a subject system. The inductive learning method they applied, requires pre-defined classes, and need a lot of effort to select and extract features and label training samples, which significantly affect the quality of learned concepts or models. Besides, their methods require high quality change reports that are not always readily available. Gall et al. [10] use information in the product release history of a system to uncover logical coupling among modules based on sequence matching. Zimmermann et al. [19] identify (heavily dependent on visualization of historical data stored in CVS archive) the fine-grained coupling between program entities like methods and fields.

Devanbu et al. [7] employed expert system and knowledge base as their underlying technology to assist in representing and deducing the relationships among components of software system. They built a system called LaSSIE, which integrates architectural, conceptual, code views of a large software system into a knowledge base represented in formal knowledge representation language and provides a semantic reasoning mechanism based on formal inference for developers to discover the structure of software system. Such knowledge-based system generally requires trained knowledge engineers to interview experts and build knowledge base and need a great effort to maintain and evolve knowledge base as system evolves.

Furthermore, they employ deduction algorithms that are computationally demanding.

2.2 Semantic UML model manipulation

There has been some work at analyzing the changes to software at the design level. Egyed [8] has investigated a suite of rule-based, constraint-based and transformational comparative methods for checking the consistency of the evolving UML diagrams of a software system. Selonen et al. [14] have also developed a method for UML transformations, including differencing. However, these projects have not explored the product of their analyses in service of evolution understanding.

3 Methodology

In this section, we present the structural modification detection algorithm, *UMLDiff*. We discuss the transaction database of class evolution histories and the Apriori algorithm used for detecting co-evolving classes given such a database. Finally, we discuss potential applications of recovered class co-evolution knowledge in the context of software maintenance.

3.1 *UMLDiff*: Class-modification detection

The overall problem of detecting and representing changes to data is important for version and configuration management. It is an active research area on its own in the area of data management. Probably the most well known algorithm for textual comparisons, *GNU diff*, was discussed as the string-to-string correction problem using dynamic programming in [17]. Used in the context of code differencing, it reports changes at the code line level rather than at higher level of abstraction of system structural modifications.

As more data and documents are stored in XML format, some sophisticated version control systems include XML-aware features to handle XML documents. The general tree-to-tree correction problem has been studied extensively [2], and has been applied to show differences between XML data [22]. However, such general tree-differencing algorithms report changes as “XML element modifications” ignoring the domain-specific semantics of the nodes. Let us consider XMI, the XML Metadata Interchange for UML models, as an example. When a class implements a new interface, a general XML-differencing tool would only report that a set of XML nodes were inserted but would not recognize the implementation of a new interface, since it does not understand the XMI semantics. For the same reason, if an attribute or a method were moved from one class to another, the change would most likely be reported as two separate activities of node addition and the deletion.

Recognizing changes while taking into account the UML-specific semantics of XMI documents is exactly the purpose of *UMLDiff*. Relying on the semantics of the model data, it identifies “moves” by hypothesizing

correspondences between additions and deletions of similarly-named elements of the same type. In the context of software evolution, where local transformations, such as refactoring, frequently involve moving features from one class to another, recognizing such “moves” is essential. As one of the most elementary operation of refactoring, a “move” often represents the redistribution of information or the reorganization of the class hierarchy, frequent perfective-maintenance modifications, such as, for example, moving methods from classes suffering strong coupling. Figure 1, discussed below, shows an example of such “move” operations.

In general, the problem of detecting the class-model changes between two snapshots of an object-oriented system can be viewed as a graph-difference problem, since class models can be viewed as specific types of directed graphs. This problem is NP-complete which makes an automatic approach impractical. Therefore, we have limited our initial exploration to considering only the inheritance hierarchies of the class model. *UMLDiff* is essentially a domain-specific tree-differencing algorithm, aware of the UML semantics captured by the XMI syntax. It takes as input two UML class models, corresponding to two snapshots of the system under analysis, represented in XMI. Such class models can be either produced in the software-design phase by the system developers, or they can be reverse engineered from the system code, using any of the currently available software engineering tools [24].

The first step of the algorithm is to parse the input forests of class models into two labelled tree structures, in which the tree nodes are labelled with the type of objects, such as class, method, etc., and their corresponding attributes, such as modifiers, data type, parameter list, etc. The target representation contains the application classes and interfaces, their fields, their methods and their inheritance, implementation, and nested class relations. Nested classes of a particular class are enclosed in a special element in the context of the containing class to distinguish them from its subclasses. Multiple-inheritance is handled by duplicating the class node (not including its children) under each of its super classes.

The next step of the algorithm is to identify the *after-before* changes between the two tree structures, in terms of the “additions”, “removals”, “moves”, “renamings” and “signature changes” of super- and sub- classes, interfaces, and their fields and methods. Currently, the comparison is based on simple identifier matching of the signatures of the various object-oriented entities of the same type.

This UML differencing process brings to the surface structural modifications to the software design from one snapshot to another. The results are represented as *change trees*, i.e., trees of structural modifications, which, if applied to the *before* version would result in the *after* version. Change trees are represented in an XML-based syntax and are visualized to the user as shown in Figure 1.

The different icons to the left of each node represent the different object-oriented entities: “class”, “interface”, “method”, and “field”. The top-right adornments show the modifiers of the object, for example, “abstract”, “static”, etc. The bottom-right adornments represent the status of a particular object. It can be plus sign for “add”, minus sign for “remove”, filled triangle for “rename”, empty triangle for “change signature”, arrow with minus sign for “move out from source”, arrow with plug sign for “move into target”. Figure 1 shows that a new abstract class, “Statement”, was created with three newly created abstract methods, “eachRentalString”, “footerString”, and “headerString”. The “value” methods of its two subclasses, “HTMLStatement” and “PlainStatement”, were pulled up into the new class “Statement”. This change tree corresponds to the differences between version 27 and 28 of the extended refactoring sample from Fowler’s book [9] as found in [20]. It represents the modifications to the class model after an “Extract Superclass” refactoring, which is described as follows: “if you have two classes with similar features, then create a superclass and move the common features to the superclass [9]”

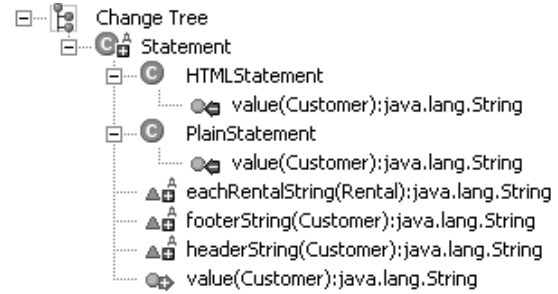


Figure 1 An example of change tree

3.2 The detection of co-evolving classes

UMLDiff reports the structural changes between two snapshots of a system’s class models. There are N such models in an evolving software system with N successive snapshots, and consequently *UMLDiff* can be applied $N-1$ times to generate the differences between the $(I+1)^{\text{th}}$ and I^{th} versions, where $1 \leq I < N$. Thus, $N-1$ change trees can be obtained that record the structural modifications, in terms of the “additions”, “removals”, “moves”, “renamings” and “signature changes” of classes, interfaces, and their fields and methods, when software evolves from one snapshot to another.

We think of a change tree, plus the first snapshot, as a transaction that records the system classes that have been modified (including creation and deletion) in the corresponding snapshot. For a software system with N snapshots, a database with N transactions is generated that describes the class level evolution of software system. Each transaction T contains a set of classes with a unique identifier, the snapshot *ID* (*SID*). Table 1 shows a hypothetical transaction database. We assume that the

system, in its final version, has five classes C1, C2, C3, C4, C5. Its first snapshot contains only C1, C2, C5 and in the next version, class C2 is modified and class C4 is added, and so on.

Table 1 Transaction database for class evolution

SID	Set of classes
S01	C1, C2, C5
S02	C2, C4
S03	C2, C3
S04	C1, C2, C4
S05	C1, C3
S06	C2, C3
S07	C1, C3
S08	C1, C2, C3, C5
S09	C1, C2, C3

We apply the Apriori association-rule mining algorithm (we used its implementation in the Weka [25] toolkit) to discover co-evolving classes, in most cases previously undocumented or unknown, that have common change behaviors. We describe briefly the Apriori algorithm here. Interested readers are referred to [1] for more details.

Given a set of transactions, the original Apriori algorithm generates all association rules with at least some user-specified *minimum support* and *confidence*. The algorithm involves two subproblems. First, generate all sets of items (itemsets) that have transaction support above minimum support. The *support* for an itemset is the number of transactions that contain the itemset. Itemsets with minimum support are called *large itemsets* and all others are *small itemsets*. Next, the large itemsets are used to generate the desired rules. The general idea is that, if ABCD and AB are large itemsets, then the rule $AB \Rightarrow CD$ holds if its *confidence*, i.e., the ratio $\text{support}(ABCD)/\text{support}(AB)$ is greater than the user-specified minimum confidence. Note that the rule will surely have minimum support because ABCD is large.

However, the so-called strong rules generated by support-confidence framework may not be interesting to the user, since the antecedent and consequent may be negatively associated, which means the occurrences of one of them may decrease the likelihood of the occurrence of the other. The alternative measure *lift* can be used to measure the statistical dependence (correlation) between the occurrences of itemsets. The Weka toolkit also support support-lift framework, which we are using to generate correlation rules.

For the transactional data shown in Table 1, if the minimum support is set to 20% and the minimum confidence to 40%, then we can generate the rules $C1 \wedge C2 \Rightarrow C5$ or $C5 \Rightarrow C1 \wedge C2$, which indicate a co-evolution relation among these three classes. Note that the rule allows a consequent to have more than one item, which is the advantage by applying association rule method over classification method.

3.3 Understanding co-evolution in the context of software maintenance

In this section, we discuss how uncovered co-evolution relations may assist software engineers in their task of understanding software evolution and planning future maintaining activities.

3.3.1 Intentional co-evolution

Class co-evolution may be the consequence of the original design and implementation decisions. For example, an approved design may require certain classes to be modified for every new feature addition. Unfortunately, such design intent is not always documented; the software developers just “know” that they have to modify a certain set of classes when make a certain kind of change. However, such knowledge is easily lost with staff turnover.

In this sense, detecting co-evolving classes can be thought of as a design recovery process that provides a way to identify high-level relations among classes, which increases the overall understanding of a long-lived evolving system. This co-evolution relation can be used as the basis for advice on maintenance activities. For example, if three classes were often changed together, when a developer modifies two of them, it would be recommended to examine if a change is also necessary to the third one.

3.3.2 “Maintenance smells”

Fowler [9] describes the “when” of refactoring in terms of smells where suggest the possibility of refactoring. Some of them can be characterized as “evolution smells”, which are not obvious in a single snapshot of system but can be identified by analyzing changes made to system over time. Two examples of such evolution smells are the following:

- **Shotgun Surgery:** whenever a kind of change is made necessitating many little changes to many different classes.
- **Parallel Inheritance Hierarchies:** whenever subclassing one class results in having to subclass other classes (a special case of shotgun surgery).

The most observable evolution characteristic of these smells is the classes that have been changed together over time. Therefore, the identification of co-evolving classes may help software engineers discover whether the system suffers from these smells. For example, the original design of a software system may have followed the Model/View/Controller (MVC) model. However, due to side effects of changes that the system has gone through over its life span, a cluster of classes, belonging in the presentation layer and the data-model layer, are discovered to evolve in parallel. That may reveal the high coupling between presentation and data model layer, which means that the current system implementation deviates from the original design intent. A “Separate presentation from data

model” refactoring could be applied to improve the cohesion and reduce the coupling.

3.3.3 System instability

In addition to enabling maintenance advice and providing evidence of “smells” necessitating particular refactorings, class co-evolution may also be used as an indicator of general “system instability”.

Bianchi et al. [5] and Hassan et al. [11] claim that the entropy of a software system is a good indicator of the degree of disorder of its structure. The term “entropy” refers to the amount of uncertainty related to information in a distribution. Intuitively, in the context of software evolution, if a software system is being modified across all its modules, it will have highest entropy, and the software maintainers will have a hard time keeping track of all the changes. Both researches rely on maintenance documentation to determine the relations among system components.

In a similar vein, Bevan [4] defines software instability as a set of related artifact elements that have often changed together. She uses a static dependence graph to visually identify such related software artifacts. We believe that the co-evolving classes detected by applying association rule mining could provide a good primary input for system instability analysis. We plan to evaluate the overall development process by analyzing the knowledge revealed by co-evolving classes. We expect to be able to identify abnormal phases of software evolution due to class co-evolution.

4 The case study

The overall objective of our design-level evolution analysis work is to support software practitioners to understand software evolution at a higher level of abstraction by automatically identifying and analyzing the evolution characteristics of system and its components. In this section, we report on a case study we have conducted in order to assess our class co-evolution method.

Mathaino [12] is a research prototype tool that can be used to migrate text-based legacy interfaces to modern web-based platforms. It underwent 91 builds from July 2000 till February 2001. The first version has 29 classes, 284 methods, and 256 attributes. The last version has 144 classes and about 1800 methods and 1800 fields. We reverse-engineered Mathaino to generate 91 class models from its source code versions and run *UMLDiff* to surface the structural modifications when software evolves from one version to another.

By applying Apriori mining, we discovered several co-evolving class clusters, for example, (a) “AbstractForm”, “AbstractInputField” and “AbstractOutputField”, and (b) “FormNavigator”, “MathainoXHTMLGUITranslator” and “TaskDataExtractor”. Note that the classes in the first cluster share the same prefix or suffix; the developer of Mathaino followed a principled notation convention,

which might enable the discovery of the co-evolution relation through code inspection. At the same time, given the naming convention, the prefix similarity may indicate that the co-evolution is intentional. This is not the case for the second cluster, however. It might also be intentional co-evolution. But it cannot be recovered by simply checking class names.

By analyzing the evolution of the first 18 versions, we also discovered that a set of classes, “MathainoCreatePlugin”, “MathainoParserPlugin”, and their subclasses were originally co-evolving, but not after the 20th version. The names of these classes suggest that there may exist a parallel inheritance hierarchy in the system. In the 19th version, we identified the instance of “Extract Superclass” refactoring that involves the newly created ancestor abstract class “MathainoPlugin”, which suggests that the developer of Mathaino probably made some structural changes to remove this co-evolving in the 19th version. His report [16] validated our intuition based on the analysis of evolution of these classes. Until its 18th version, Mathaino had two separate plugin hierarchies, one for the “creator” and the other for the “parser”, and two separate plugin loaders and registries respectively. This design was not flexible enough to handle new types of interactions and would easily result in “parallel inheritance”. At the code level, a lot of code was duplicated. It would seem then that the Mathaino developer noticed the problem and made a design decision in the 19th version to “Extract Superclass” from these two separate plugin class hierarchies, and their corresponding plugin loaders and registries, which reduced the code duplication and made the system architecture much more maintainable.

5 Conclusion and future work

In this paper, we discussed our method for detecting clusters of two or more co-evolving classes in an object-oriented software system. The method relies on readily available data, as opposed to consistently documented software project change requests. It takes as input a sequence of class models of the system represented in XMI, reverse-engineered from a corresponding set of code versions. These models are compared using the *UMLDiff* algorithm to detect various types of changes to the system’s classes, interfaces, and their fields and methods. Finally, the extracted class-evolution histories are mined, using Apriori, to extract association rules indicating class co-evolutions.

We have discussed three potential applications of class co-evolution discovery in the context of software maintenance: advice regarding the scope of future maintenance activities, guidance for particular refactorings and, potentially, recognition of system instabilities. In our Mathaino case study, we discovered several class co-evolution instances and we also found evidence that the project developer acted according to the advice that our

theory would have generated, had it been in place during the system's development.

The approach of detecting co-evolving classes presented in this paper has been implemented as a part of one of two analysis plugins in Eclipse [21], in the context of the JReflEX project [18]

For the future work, we are investigating whether a more specific notion of co-evolution, in terms of the specific modifications identified by *UMLDiff*, would enable more precise maintenance and refactoring advice. The more specific co-evolution notion would enable us to answer such question as "Are there any classes in some part of the hierarchy that are often restructured when some classes are added into another part of the hierarchy?"

We also plan to conduct a similar case study on a much more complex software system, Eclipse [21], which is built on an extensible plugin framework. The core of Eclipse has more than 60 plugins, most of which have several dozens of revisions. Its core plugins have been divided into several subgroups, such as compare support, team support, search, user interface, etc., which have been developed in separate IBM research branches. More important, it is an "active" project in contrast to the "closed" case study, discussed here. Eclipse will provide a good test bed to evaluate the scalability, "on-going" usage and effectiveness of our method.

Acknowledgment

This work was supported by an Eclipse Innovation Grant and by CSER, the Consortium for Software Engineering Research, and NSERC the National Sciences and Engineering Research Council of Canada.

References

1. R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules", *Proceedings of the 20th International Conference on Very Large Databases*, September 1994.
2. D. Barnard, G. Clarke and N. Duncan, "Tree-to-tree Correction for Document Trees", Technical Report 95-375, Queen's University, January 1995.
3. B. Bellay and H. Gall, "An evaluation of reverse engineering tool capabilities", *Journal of Software Maintenance: Research and Practice*, 1998, 10(5):305-331.
4. J. Bevan and E.J. Whitehead, "Identification of software instabilities", *Proc. of the 10th Working Conference on Reverse Engineering*, 2003, pp. 134-143.
5. A. Bianchi, D. Caivano, F. Lanubile, and G. Visaggio, "Evaluating software degradation through entropy", *Proceedings of the 11th International Software Metrics Symposium*, 2001, pp. 210-219.
6. E.J. Chikofsky and J.H. Cross, "Reverse engineering and design recovery: A taxonomy", *IEEE Software*, January 1990, pp 13-17.
7. P. Devanbu, R.J. Brachman, P.G. Selfridge and B.W. Ballard, "LaSSIE: A knowledge-based software information system" *Communications of the ACM*, May 1991, 34(5):35-49.
8. A. Egyed, "Scalable Consistency Checking between Diagrams - The VIEWINTEGRA Approach," *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, San Diego, USA, 2001.
9. M. Fowler, "Refactoring: Improving the Design of Existing Code", Addison-Wesley, 1999.
10. H. Gall, K. Hajek and M. Jazayeri, "Detection of Logical Coupling Based on Product Release History", *Proceedings of the International Conference on Software Maintenance*, Bethesda, Washington D.C., November 1998.
11. A.E. Hassan and R.C. Holt, "Studying the chaos of code development", *Proc. of the 10th Working Conference on Reverse Engineering*, 2003, pp. 123-133.
12. R. Kapoor and E. Stroulia, "Mathaino: simultaneous legacy interface migration to multiple platforms", *Proceedings of 9th International Conference on Human Computer Interaction*, 2001.
13. K.A. Kontongiannis and P.G. Selfridge, "Workshop report: The two-day workshop on research issues in the interaction between software engineering and artificial intelligence" *Automated Software Engineering*, 1995, 2:87-97.
14. P. Selonen, K. Koskimies, M. Sakkinen, "Transformations between UML Diagrams", *Journal of Database Management*, Vol. 14, No. 3, 2003.
15. J.S. Shirabad, T.C. Lethbridge, and S. Matwin, "Supporting Software Maintenance by Mining Software Update Records", *Proceedings of 17th International Conference on Software Maintenance*, Italy, 200, pp. 22-31.
16. E. Stroulia and R. Kapoor, "Metrics of Refactoring-based Development: An Experience Report", *Proceedings of the 7th International Conference on Object-Oriented Information Systems*, 27-29 August 2001, pp. 113-122, Springer Verlag.
17. R. A. Wagner and M.J. Fischer, "The string-to-string correction problem", *Journal of the ACM*, January 1974, 21(1):168-173.
18. K. Wong, W. Blanchet, Y. Liu, C. Schofield, E. Stroulia, and Z. Xing, "JReflEX: Towards Supporting Small Student Software Teams", IBM Eclipse Workshop at OOPSLA 2003.
19. T. Zimmermann, S. Diehl, and A. Zeller. "How History Justifies System Architecture (or not)", *Proceedings of International Workshop on Principles of Software Evolution*, Helsinki, Finland, September 2003.
20. <http://www.cs.unc.edu/~stotts/COMP204/refactor>.
21. Eclipse, <http://www.eclipse.org>.
22. Mosell EDM Ltd, <http://www.deltaxml.com>.
23. Rational Rose, <http://www.rational.com>.
24. Together, <http://www.togethersoft.com>.
25. Weka, <http://www.cs.waikato.ac.nz/~ml/weka>

Defining and Qualifying Components in the Design Phase

Andrew O'Fallon, Orest Pilskalns, Andrew Knight, Anneliese Andrews
School of Electrical Engineering and Computer Science
Washington State University, Pullman, WA 99164
{aofallon, orest, aknight, aandrews}@eecs.wsu.edu

Abstract

Component based development in the design phase necessitates a comprehensive look at both static and dynamic UML views. If a design is to incorporate third-party components, one must define component interfaces. We propose a method for defining components in the design phase that can be used for qualification purposes. Coupling and frequency metrics are used to make component definition decisions. Component interface definitions allow for qualifying candidate components.

1. Introduction

Software design is increasingly including component based software development [3]. This happens from two perspectives: (1) a software designer wants to use components and needs to define how components fit with the remainder of the design. Then candidate components must be evaluated whether they fit into the design. The latter is part of component qualification. (2) a software designer wants to design components for reuse. This could be as part of a product line architecture, or as part of a set of components developed for reuse.

In either case, one needs to determine the component boundaries and interfaces. Second, candidate components need to be evaluated how well they fit into the overall design. i. e. how well they fit the component boundaries.

We propose an analysis method that works for designs expressed in UML Class Diagrams and Sequence Diagrams. It is derived from Pilskalns et al. [11]. In [11], a model is derived from Class Diagrams and Sequence Diagrams that integrates both structural and behavioral characteristics of the design. It is used to generate and execute tests. This paper uses the same model to evaluate component boundaries and define components with high cohesion and low coupling. Candidate components are then qualified by how well they fit the interfaces of the component with the rest of the design.

Section 2 describes existing work on component selection and qualification. Section 3 explains the approach used

for component definition. Section 4 defines the method for component qualification. Section 5 illustrates the analysis method on an example. Section 6 draws conclusions and suggests further work.

2. Background

In [6, 7] Kontio et al. apply a selection and evaluation method to multiple case studies. The method investigated is referred to as OTSO (Off-The-Shelf Option). OTSO describes a systematic approach to selecting packaged components. The method includes six phases: search, screening, evaluation, analysis, deployment, and assessment. Lester et al. [8], apply the idea of using stereotypes, class compartments, and association rules for qualifying the reuse of software artifacts. These UML constructs are used to define search criteria for reuse candidates. The stereotype is used to limit the search of objects to those objects that contain the stereotype or are derived from the object with the stereotype. Attribute-Value classification can be used to provide a structured way to integrate association roles into the search criteria of an object.

The COTE (COmponent TEsting) project [5] is concerned with developing an integrated environment (IE) for qualifying and testing components. The research is primarily interested in using the IE for components modeled in UML. Sequence diagrams are used to generate UML test profiles.

These methods are more concerned with component evaluation and qualification than definition. Further, they are fairly high level. We see our method as a more detailed analysis approach whose results can be used in the context of an OTSO evaluation. Similarly, our component qualification method can be seen as a method for selecting candidate components that can then be tested using COTE.

3. Defining Components

We define components by (1) creating a model that merges the static and dynamic information of UML Class

and Sequence diagrams, (2) applying an operation profile to the model to collect metrics, and (3) analyzing the metrics to identify boundaries in the model for defining potential component interfaces. The first step in defining components is to convert the class diagrams and sequence diagrams into a directed acyclic graph that can be analyzed for cohesion [1] and coupling (Constrained Object Method Directed Acyclic Graph or COMDAG). We do this in two steps based on Pilskalns et al. [11]. First we convert classes into constrained class tuples (CCTs) that describe attributes, methods, and inheritance relationships of a class. A CCT is contained in each node of the COMDAG and represents an instantiated class. A set of CCTs can be used to define a component, since all of the interface information is available. The sequence diagrams are converted into graphs (COMDAG), starting with the first method call, following the paths through the sequence diagram. Nodes are defined by the method, object, and classes involved. Edges connect method sequences as specified in the sequence diagram. Table 1 shows the definition of the CCT as specified in [11]. Here we do not need all parts of this definition, since we do not need to generate and execute test cases.

The COMDAG can be constructed by mapping elements of the Sequence Diagrams to a graph. The COMDAG is a tuple $\langle V, E, s \rangle$ where V is a set of vertices, E is the set of edges, and s is the starting vertex. Each vertex, v , is defined by the triple $v = \langle o, \langle M \rangle, CCT(c) \rangle$, where o is an object, $\langle M \rangle$ is a method tuple, c is a class. An edge E , represented by the tuple $\langle v_1, v_2 \rangle$, consists of a pair of vertices that represent the ordering between vertices v_1 and v_2 defined by the sequence diagram.

Once the CCT and COMDAG are defined, it is possible to define a set of (connected) COMDAG vertices with the smallest number of connections ($\#conn$). To define and design a component and its boundaries select, a set of nodes $\langle A \rangle$ where every node is directly connected to at least one other member in the set. (e.g. a connection is defined as $\langle v_n, v_{n+1} \rangle$). Since each node contains a CCT, the newly defined component contains a complete description of the potential interface. A min-cut algorithm can be employed or a designer can manually identify potential component boundaries. In addition, we assume an operational profile has been defined for use cases; that is, each use case is associated with a (relative) frequency. Execution as in [11] or tracing a use case through the COMDAG identifies how often interfaces are activated ($\#act$). A decision on which nodes are part of a defined component are then made based on $(\#conn, \#act)$. The designer can then choose either the interface with the fewest connections, or the interface with the fewest activations. Alternatively, the designer may have set a threshold for $\#act$ and then selected the interface with the fewest $\#conn$ that falls within the threshold. The component $\langle A \rangle$ identified with this approach is a subset of

the vertices in the COMDAG. The activation and connections need not be the only metrics we use. Since the CCT contains attribute and method information, metrics can be collected for class size, method signatures, attribute types, etc. For instance, one of our criteria for a potential component may only select classes that have a maximum of ten methods. The CCTs that define our component will be used in the next section to qualify candidate components.

4. Qualification

This approach determines if an implemented candidate (COTS) component qualifies for the current design and architecture of the system. The analysis is based on comparing the interfaces of the design component as defined in the prior section with a list of implemented candidate components. Interfaces are described in terms of information about the methods, parameters, and attributes as contained in a Constrained Class Tuples (CCT), of Table 1.

Interface analysis determines if an implemented candidate component satisfies the requirements of the system. The set of attributes and methods is described in the form shown in rows 4 and 5 of Table 1, respectively. We assume that a component X may contain multiple attributes and methods, hence many different attribute and method signatures. We will define an interface as comprising multiple attribute and method signatures.

Step 1: The first step to asserting that a candidate component is sufficient for a designed component is to extract the necessary information from the signatures of both the designed and candidate components. Information must be extracted from both the method and attribute sets.

Attribute Extraction, $Attr(X)$, is a function that extracts pertinent attribute information, for component qualification from component X . Component X contains a set of attributes of the form seen in Table 1 row 4. $Attr(X)$ returns the set of all (attribute type, invariant) pairs of component X . We define a metric $\#Attr(X)$ which is equivalent to $|Attr(X)|$.

Method Extraction, $Meth(X)$, extracts pertinent method information, for component qualification from component X . Component X contains a set of methods of the form in Table 1 row 5. $Meth(X)$ returns the set of all (return type, invariant, parameter) triples of component X . We define a metric $\#Meth(X)$ which is equivalent to $|Meth(X)|$.

The method triples $Meth(X)$ and attribute pairs $Attr(X)$ of a component are called its signature.

Step 2: Next we determine if the interface of a candidate component B matches the interface of a designed component A . Each method triple and attribute pair of A is compared to each method triple and attribute pair of B . A *complete match* is found if the signature of the method or at-

Reference #	Identifier	Definition
1	CCT(class name)	$\langle class\ name, \{\{Parent\ CCT\}, \{\{Attribute\}\}, \{\{Method\}\}\rangle$
2	Attribute	$\langle a_{name}, attribute\ type, a_{invariant}, \langle CCT \rangle \rangle$
3	Method	$\langle m_{name}, return\ type, visibility, m_{invariant}, \langle Parameters \rangle \rangle$
4	$\{Attributes\}$	$\{\langle a_{name_1}, attribute_{type_1}, a_{invariant_1}, \langle CCT \rangle_1 \rangle, \langle a_{name_2}, attribute_{type_2}, a_{invariant_2}, \langle CCT \rangle_2 \rangle, \dots, \langle a_{name_m}, attribute_{type_m}, a_{invariant_m}, \langle CCT \rangle_m \rangle\}$
5	$\{Methods\}$	$\{\langle m_{name_1}, return_{type_1}, visibility_1, m_{invariant_1}, \langle Parameters \rangle_1 \rangle, \langle m_{name_2}, return_{type_2}, visibility_2, m_{invariant_2}, \langle Parameters \rangle_2 \rangle, \dots, \langle m_{name_n}, return_{type_n}, visibility_n, m_{invariant_n}, \langle Parameters \rangle_n \rangle\}$

Table 1. A constrained class tuple and its elements.

tribute in A has the same return type, invariant, and parameter tuple or attribute type and invariant of the method or attribute, respectively, in B . A *partial match* is found only if some of the elements of the method or attribute signature in A match the method or attribute signature in B . For a match we need to recursively search through A 's CCT and Parent CCT for methods and attributes whose signatures match the signatures of B (refer to Table 1 row 1).

Step 3: The third step is to determine if the interface of a candidate component B exceeds the interface of a design component A . The interface of B safely exceeds the interface A if it contains enough methods and attributes to match all signatures of methods and attributes in A . Essentially the signature of A , as defined by its attribute pairs $Attr(A)$ and method triples $Meth(A)$, must be a proper subset of the signature of B .

This helps to determine whether or not a given design that requires attributes described by component A can be satisfied by candidate component B . For example, imagine a component A which requires five cards to represent a poker hand. Each of the cards is represented as a String. If candidate component B contains six cards represented as Strings, then component A is a proper subset of component B .

Step 4: We determine a qualification measure. We need to perform operations on sets of attributes and methods of components in order to determine if a candidate component satisfies the requirements. We perform weighted operations on both the attribute and method sets of components. The attribute sets are weighted w_a and the method sets are weighted w_m according to the Analytic Hierarchy Process (AHP) [12]. In the following sections, the fitnesses and coverages computed for both attribute and method sets are also weighted by AHP.

Given two components A and B , with attributes $\{aa_1, aa_2, aa_3, \dots, aa_m\}$ and $\{ba_1, ba_2, ba_3, \dots, ba_n\}$ respectively, the *difference* between component A 's and component B 's attributes is defined in equation 1.

$$\#attrOver(B, A) = \#Attr(B) - \#Attr(A) \quad (1)$$

The difference in equation 1 is the number of attribute elements in $Attr(B)$, but not in $Attr(A)$.

The difference between two components and their attributes is considered the *attribute overhead* of the component. For example the design of a poker game needs a component A which has five cards all represented by Strings, $\{card_1, card_2, card_3, card_4, card_5\}$. Consider a candidate component B which contains six cards all represented by Strings, and a game type represented as an integer number, $\{card_1, card_2, card_3, card_4, card_5, card_6, game_type\}$. Applying $\#Attr(B) - \#Attr(A)$ to the two components results in $\#attrOver = \{card, game_type\}$. The difference in this case represents the *attribute overhead* of the components. Representing the attribute overhead as a percentage is defined, refer to equation 2.

$$attrOver\% = \frac{\#attrOver}{\#Attr(B)} * 100 \quad (2)$$

For this example the percentage of overhead is $2/7 = 29\%$.

Given two components A and B , with methods $\{am_1, am_2, am_3, \dots, am_m\}$ and $\{bm_1, bm_2, bm_3, \dots, bm_n\}$ respectively, the difference between component A 's and component B 's methods is defined in equation 3.

$$\#methOver(B, A) = \#Meth(B) - \#Meth(A) \quad (3)$$

The method difference is the number of method elements in $Meth(B)$, but not in $Meth(A)$. The difference between two components and their methods is considered the *method overhead* of the component. For example the design of a poker game needs a component A which has a deal and shuffle method, $\{getHand(), shuffle()\}$.

If a candidate component B is available which contains methods for `getHand()`, `shuffle()`, and `getBet()`, $\{getHand(), shuffle(), getBet()\}$, then applying $\#Meth(B) - \#Meth(A)$ to the two components results in $\#methOver = \{getBet()\}$. The method overhead of the components as a percentage is defined in equation 4.

$$methOver\% = \frac{\#methOver}{\#Meth(B)} * 100 \quad (4)$$

For this example the percentage of overhead is $1/3 = 33\%$.

The *intersection* of two components A and B , with attributes $\{aa_1, aa_2, aa_3, \dots, aa_m\}$ and $\{ba_1, ba_2, ba_3, \dots, ba_n\}$ respectively, is defined in equation 5.

$$\#attrInt = |Attr(A) \cap Attr(B)| \quad (5)$$

The attribute intersection is the number of attribute elements in both $Attr(A)$ and $Attr(B)$.

The intersection between two components indicates coverage. Thus all attribute elements that are present in $Attr(A)$ and in $Attr(B)$ represent the *attribute coverage*. Attribute coverage can be calculated as in equation 6.

$$attrCov\% = (1 - \frac{\#Attr(A) - \#attrInt}{\#Attr(A)}) * 100 \quad (6)$$

For example, the design of a poker game needs a component A which has five cards all represented by Strings, $\{card_1, card_2, card_3, card_4, card_5\}$. If a candidate component B is available which contains six cards all represented by Strings, and a game type represented as an integer number ($\{card_1, card_2, card_3, card_4, card_5, card_6, game_type\}$), then applying $Attr(A) \cap Attr(B)$, results in $I = \{card_1, card_2, card_3, card_4, card_5\}$. Thus, $|Attr(A)| = 5$ and $\#attrInt = 5$, which indicates that the coverage is $1 - (5 - 5) = 100\%$. However, the attribute coverage must be weighted by w_{ac} . The *attribute fitness* of the component is defined in equation 7.

$$attrFit\% = (1 - \frac{attrOver\%}{100}) * 100 \quad (7)$$

The attribute fitness is $1 - 0.29$ or 71% . The attribute fitness must also be weighted by w_{af} .

The intersection of two components A and B , with methods $\{am_1, am_2, am_3, \dots, am_m\}$ and $\{bm_1, bm_2, bm_3, \dots, bm_n\}$ respectively, is defined in equation 8.

$$\#methInt = |Meth(A) \cap Meth(B)| \quad (8)$$

Where method intersection is the number of method elements in both $Meth(A)$ and $Meth(B)$.

The intersection between two components indicates coverage. Thus all method elements that are present in $Meth(A)$ and in $Meth(B)$ represent the *method coverage*. The method coverage is calculated as in equation 9.

$$methCov\% = (1 - \frac{\#Meth(A) - \#methInt}{\#Meth(A)}) * 100 \quad (9)$$

For example, the design of a poker game needs a component A which has a deal and shuffle method, $\{getHand(), shuffle()\}$. If a component B is available which contains methods for `getHand()`, `shuffle()`, and `getBet()`, $\{getHand(), shuffle(), getBet()\}$, then applying $Meth(B) \cap Meth(A)$ to the two components results in $I = \{getHand(), shuffle()\}$. Thus, $|Meth(A)| = 2$ and $\#methInt = 2$, which indicates that the method coverage is $1 - (2 - 2) = 100\%$. However, the method coverage must be weighted by w_{mc} . The *method fitness* of the component is defined in equation 10.

$$methFit\% = (1 - \frac{methOver\%}{100}) * 100 \quad (10)$$

The method fitness is $1 - 0.33$ or 67% . The method fitness must also be weighted by w_{mf} .

A component B *properly satisfies* a component A if and only if the fitness and coverage is x for both attributes and methods. A component B *satisfies* a component A if and only if the fitness and coverage is at least y for both attributes and methods, where $x > y$ and $x \leq 50\%$. If the fitness and coverage for both attributes and methods is less than y , then a different component should be considered. The sum of attribute fitness and attribute coverage should be as close to 100% as possible to ensure that a given component *attribute qualification* satisfies the requirements of the system. Also, the sum of method fitness and method coverage should be as close to 100% as possible to ensure that a given component *method qualification* satisfies the requirements of the system. Adding the results of the attribute qualification and the method qualification together and dividing by 2, results in the *overall qualification* of the component for the system.

5. Example: Analysis & Qualification

We created a UML designed application to demonstrate the component definition and qualification processes. The application creates a two-dimensional convex polygon filled with either a solid color or Gouraud shading. The inputs to the program are two filenames: an input file and an image file. The input file contains the size, a list of coordinates,

Use Case: Create Polygon
Intent: Create polygon image from input file
Pre Conditions: Input file is correctly specified
Post Conditions: Polygon image file created
Description:
 1. User executes program with 2 command line arguments
 2. System returns polygon image file

Figure 1. Polygon Use Case

and color values. The image file can have five different formats: BMP, GIF, JPG, PNG, or a TIF.

The original class diagram contained 14 classes. Figure 2 shows a simplified version, with many classes hidden behind the image interface. The sequence diagram, Figure 3, has been simplified as well. Both diagrams were designed using UML 2.0 [9]. The COMDAG, Figure 4, was derived from the sequence diagram. The loops in the COMDAG have been retained to reduce the size, but in practice they would be unraveled into a directed acyclic graph.

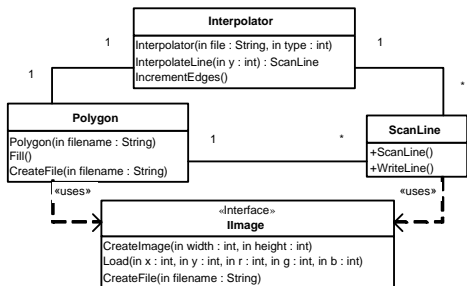


Figure 2. Polygon Class Diagram

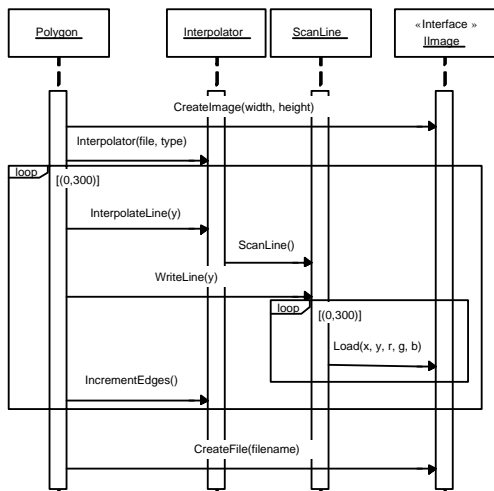
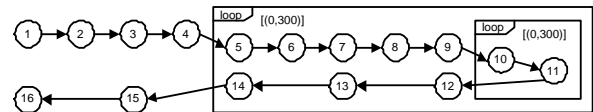


Figure 3. Polygon Sequence Diagram

In our definition process we use an Operational Profile consisting of Use-Cases and their frequency of execution. For this example, we use one of the primary Use-Cases, see Figure 1, to simulate execution. There are three nodes directly connected to the Image Component: 1, 10, and 15. Despite only 3 connections, when we simulate the Use-Case

there are 90,002 activations. Obviously the boundaries to the image interface should also be our component boundaries. Therefore, our component is defined in terms of the CCT's describing image interface.



1. < p, < CreateImage, < 300, int >, < 300, int > >, Polygon >
2. < IImage, < return >, IImage >
3. < p, < Interpolator, < filename, string >, < type, int > >, Polygon >
4. < i, < return >, Interpolator >
5. < p, < InterpolateLine, < y, int > >, Polygon >
6. < i, < ScanLine, null >, Interpolator >
7. < s, < return >, Scanline >
8. < i, < return >, Interpolator >
9. < p, < WriteLine, < y, int > >, Polygon >
10. < s, < Load, < x, int >, < y, int >, < r, int >, < g, int >, < b, int > >, ScanLine >
11. < IImage, < return >, IImage >
12. < s, < return >, ScanLine >
13. < p, < IncrementEdges, null >, Polygon >
14. < i, < return >, Interpolator >
15. < p, < CreateFile, < filename, string, [Type = bmp, gif, jpg, png, tif] > >, Polygon >
16. < IImage, < return >, IImage >

Figure 4. Polygon COMDAG

The purpose of the Image Component is to provide a way for the program to write an image file. We need to be able to specify the image size, the RGB value for each pixel, and create one of five file types. While the tasks the component should perform are quite simple, the component functionality may not conform to our definition. A third party image component may not allow us to create the desired file types, or the interface for loading information may be non-compliant. Other complications arise if the third party component supports extra operations that our program does not use. Although extra functions can be useful, when unused, the excess overhead increases code size without increasing effectiveness.

	Attr (P)	Attr (IIC)
Position	int x, int y	int x, int y
Color	int r, int g, int b	int r, int g, int b, int a
File Type	bmp, gif, jpg, png, tif	bmp, gif, jpg, png, tif, pbm, pgm, ppm, tga

Table 2. Attributes Required vs. Provided

We will apply Set Analysis to qualify the Imaginary Image Component (*IIC*), a COTS component. *IIC* runs on the Java platform, and the interface is provided as a class. The component allows the specification of a generic file, which can be saved as nine different types. *IIC* allows the image file to be changed one pixel at a time through the specification of a Position and a Color. Although, neither the position nor color class are used in the Polygon specification, we can use CCT's to unravel each structure down to their base types, and perform the comparisons there. Table 2 shows the unraveled attributes. After the image has been created, *IIC* provides a variety of methods to alter the image

Meth (P)	Meth (IIC)
CreateImage(int width, int height)	SpecifyImage(int w, int h)
Load(int x, int y, int r, int g, int b)	LoadPixel(Position p, Color c)
CreateFile(String filename)	CreateFile(String name)
	<i>Rotate(int degree)</i>
	<i>Flip()</i>
	<i>Scale(int percentage)</i>
	<i>StretchWidth(int percentage)</i>
	<i>StretchHeight(int percentage)</i>
	<i>Crop(Position tl, Position br)</i>
	<i>Invert()</i>

Table 3. Methods Required vs. Provided

appearance. Table 3 shows a complete list of methods in the IIC component.

Everything required by Polygon that is provided by IIC is listed in normal font. By examining Table 2 & 3, it is also apparent that IIC is *sufficient* to cover Polygon's requirements. Because IIC is sufficient, coverage is 100% for both attributes and methods.

$$\#AttrInt = |Attr(P) \cap Attr(IIC)| = 10$$

$$AttrCov\% = (1 - \frac{\#Attr(P) - \#AttrInt}{\#Attr(P)}) \cdot 100 = 100\%$$

$$\#MethInt = |Attr(P) \cap Attr(IIC)| = 3$$

$$MethCov\% = (1 - \frac{\#Meth(P) - \#MethInt}{\#Meth(P)}) \cdot 100 = 100\%$$

Although IIC is sufficient, only 2 methods have a *complete match*. LoadPixel is only a *partial match* because of the extra alpha attribute in the color parameter. All other portions of IIC *safely exceed* Polygon's requirements. The overhead is the portion of IIC that safely exceeds Polygon's requirements. In Table 2 & 3, the overhead is denoted with italics.

$$AttrOver\% = \frac{\#Attr(IIC) - \#Attr(P)}{\#Attr(IIC)} = 33\%$$

$$AttrFit\% = (1 - \frac{AttrOver\%}{100}) = 77\%$$

$$MethOver\% = \frac{\#Meth(IIC) - \#Meth(P)}{\#Meth(IIC)} = 70\%$$

$$MethFit\% = (1 - \frac{MethOver\%}{100}) = 30\%$$

The component qualification is determined by averaging the coverage and overhead together. In this case the overall qualification is 76.75%, making IIC fairly qualified as a component for the polygon program.

$$Qual = \frac{AttrCov + AttrFit + MethCov + MethFit}{4} = 76.75\%$$

6. Conclusion

This paper showed how to define a design component as part of a UML design. It also defined a method to use the component interface definition to qualify candidate compo-

nents and compute fitness metrics for the degree of fit. An example illustrates how the method works.

Since the method is based on information that can be extracted automatically from a design, it is possible implement a tool that can help in component definition and qualification. This could become a valuable design aid. Automation is our next step.

References

- [1] J. Bieman, L. Ott, "Measuring Functional Cohesion", IEEE Trans. Software Eng., vol. 20, no. 8, pp 644–657, Aug. 1994
- [2] M. Fowler, K. Scott, *UML Distilled Second Edition*, Addison–Wesley, 2000.
- [3] G. Heineman, W. Councill, *Component–Based Software Engineering: Putting the Pieces Together*, Addison–Wesley, Boston, MA, 2001.
- [4] S. Henninger, "Supporting the Construction and Evolution of Component Repositories", Proceedings of the 18th International Conference on Software Engineering, pp. 279–288, March 25–29, 1996.
- [5] C. Jard, S. Pickin, "COTE - Component Testing Using the Unified Modelling Language", ERCIM News, No. 48, pp 49–50, Jan 2002.
- [6] J. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection", Proceedings of the 18th International Conference on Software Engineering, pp. 201–209, March 25–29, 1996.
- [7] J. Kontio, S-F Chen, K. Limperos, R. Tesoriero, G. Caldiera, and M. Deutsch, "A COTS Selection Method and Experiences of Its Use", 20th Software Engineering Workshop, NASA Software Engineering Laboratory, Greenbelt, MD, 1995.
- [8] N. G. Lester, F.G. Wilkie, and D.W. Bustard, "Applying UML Extensions to Facilitate Software Reuse", UML '98 Beyond the Notation – International Workshop, pp 393–405 1998.
- [9] Object Management Group, "UML 2.0 Draft Specification", <http://www.omg.org/uml>, 2003.
- [10] S. Pickin, C. Jard, T. Heuillard, J. Jezequel, and P. Desfray, "A UML-Integrated Test Description Language for Component Testing", Proceedings PUML Workshop 2001, pp 208–223, 2001.
- [11] O. Pilskalns, A. Andrews, R. France, S. Ghosh, "Rigorous Testing by Merging Structural and Behavioral UML Representations", Proceedings UML 2003, Oct 20–24, pp 234–248, 2003.
- [12] T. Saaty, *The Analytic Hierarchy Process*, New York, NY, McGraw–Hill, 1980.

Digging into the Visitor Pattern

Fabian Büttner, Oliver Radfelder, Arne Lindow, Martin Gogolla

University of Bremen, Computer Science Department

E-mail: {green,radfelde,lindow,gogolla}@tzi.de

Abstract

In this paper we present an alternative to the VISITOR pattern, DYNAMIC DISPATCHER, that can be applied to extend existing software in a nonintrusive way, and which simulates covariant overriding of visit methods. It allows to express polymorphic operations through visitor classes in a more natural way than the original VISITOR pattern. Our solution DYNAMIC DISPATCHER can be applied without touching existing domain classes. Therefore, it is especially useful to extend frameworks and libraries. We have implemented DYNAMIC DISPATCHER as a small framework in Java and conducted performance measurements which show that the overhead should be acceptable in many real world scenarios.

1. Introduction

In the area of software development, agile methodologies like Extreme Programming [3], the Unified Process [12], and others have evolved and lots of projects are done based thereon. These approaches consider analysis, design, implementation, and testing as parallel activities. As a consequence, software design artifacts change continuously. This effect is intended to achieve a design which is constantly adequate to the emerging problem domain.

Design patterns [11] are instruments to obtain a robust, maintainable, and extensible design. One general intention of design patterns is to decouple individual concerns, so that as many parts as possible of a design remain stable according to requirement changes.

One of the most controversially discussed patterns is the VISITOR pattern. The general intention of the pattern is to allow defining operations separated from the classes they operate on. Several problematic properties of VISITOR are mentioned in the literature, for example in [13, 15]. We will target two problems especially: First, applying the pattern to existing software is very intrusive. Thus, it often cannot be used to extend existing software, particularly frameworks, as discussed in [19]. Second, in mainstream object-oriented programming languages like Java, C++, C#,

and others that do only support invariant method overriding [4, 7], VISITOR cannot express specialization. Therefore, VISITOR is not able to extract inherited operations into Visitor¹ classes.

In this paper we present an alternative to the VISITOR pattern, DYNAMIC DISPATCHER, that can be applied to extend existing software in a nonintrusive way, and which simulates covariant overriding of visit methods. It allows developers to express polymorphic operations through visitor classes in a more natural way than the original VISITOR pattern. DYNAMIC DISPATCHER can be applied without touching existing domain classes. Therefore, it is especially useful to extend frameworks and libraries.

To demonstrate that it is easily realizable, we have implemented DYNAMIC DISPATCHER as a small framework in Java using several strategies. Since we achieve the dynamic dispatching by executing additional code, there is the danger of a substantial performance impact. We conducted performance measurements which show that the relative overhead should be acceptable in many real world scenarios.

This paper is structured as follows: Section 2 introduces a simple design example, on which we apply the VISITOR pattern. In Sect. 3, we discuss the problems of VISITOR that motivate our variant DYNAMIC DISPATCHER, which is explained in Sect. 4. In Sect. 5, we present how DYNAMIC DISPATCHER can be implemented as a framework in Java. Section 6 shows the results of the performance measurements we conducted with this framework. Finally, in Sect. 7, we make a conclusion and present future work.

2. Graphics Example

In the following we will present the design of a simplified, hypothetical vector graphics software on which we will discuss some problems of VISITOR. The UML class diagram in Fig. 1 shows the static structure of the example: a picture can be composed of shapes, which can be lines, arrows, and other pictures. The reader may notice this reflexive relationship as the COMPOSITE design pattern. Two

¹We use the following conventions: ‘VISITOR’ refers to the pattern, ‘Visitor’ refers to the class in the pattern, and ‘visitor’ refers to the general concept.

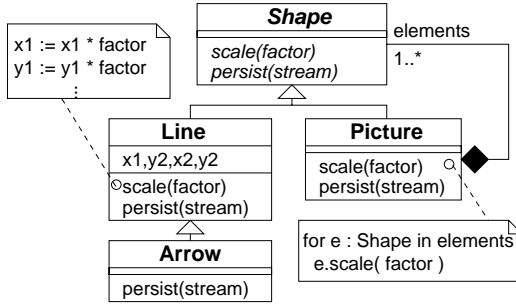


Figure 1. Graphics example

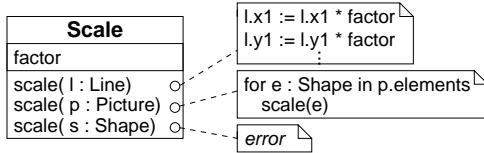


Figure 2. Extracted operation (not working)

operations are defined for shapes: *scale* and *persist*. Both operations are recursive as they descend into the tree of elements formed by the composition. For example, *scale* is applied to all elements within a picture, which may be pictures themselves, and so on.

Both shape operations are redefined (overridden) in the child classes Line and Picture. While class Arrow overrides *persist* it does inherit *scale*. Therefore, the implementation language is required to late-bind method invocation to correctly call *scale* and *persist* for lines, arrows, and pictures. Java, C++, C#, and most other static typed OO programming languages support this kind of method binding².

This is common object-oriented programming style. However, in certain situations the software developer may want to separate behavior from state, i.e., to define (some) operations outside the domain classes Shape, Line, Arrow, and Picture. Although this conflicts with common understanding of object-oriented programming in a certain sense, there are at least two recurring motivations in software engineering that justify this violation: One is to keep domain classes independent from operations in order to enable their reuse and to keep them stable when operations change. The other motivation is the use of libraries or frameworks. Since libraries typically cannot be changed, there is no alternative to defining new operations some other place. In general, this place is another class.

For example, in our graphics software the *scale* operation could be separated from our domain classes into a class Scale, as depicted in Fig. 2. The class Scale consists of three

²We use ‘method’ to refer to an implementation, otherwise we use ‘operation’

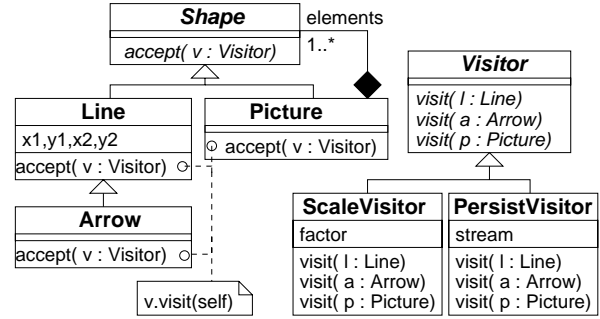


Figure 3. Graphics example - VISITOR applied

scale methods, one for each occurrence of *scale* in the domain classes before. One needs an instance of this class to scale shapes.

Unfortunately, this code does not work in mainstream object-oriented programming languages. The method *scale(l : Line)* is never called from within *scale(p : Picture)*. Instead, *scale(s : Shape)* is executed regardless of the runtime type of the elements in the picture. That is because these languages do not support double-dispatch of method invocations. Instead, the method to be invoked is chosen solely based on the runtime type of the receiver (self). If we had omitted *scale(s : Shape)*, Scale would not even compile. *scale* is only overloaded and the call *scale(e)* is statically bound to a method according to the argument’s reference type. An in-depth comparison of the different late-binding signatures of Java, C++, C# and others can be found in [4].

A common solution to this problem is the VISITOR pattern. The basic idea is to call an (invariantly) overridden method *accept* in the class hierarchy you work on. In turn, this method calls the correct *visit* method (as *persist* is now named) on the caller. Figure 3 shows the static structure of our graphics example after applying VISITOR. The *accept* method always consists of the same piece of code: *visit(self)*. Consequently, the early-bound call to the overloaded method *persist* is replaced by a late-bound call to the overridden method *accept*. The type of *self* is always the type of the class, therefore it is always different and the correct overloaded method is selected in the caller at compile-time. Some people append the type name to Visitor methods, i.e. *visitline*, *visitarrow*, and so on to make this fact clearer, but there is no difference at this point between overloading and renaming. Since nothing special happens in the *accept* method, we can reuse it for all operations we want to move out of the class hierarchy, like *scale*, *persist*, etc. The classes that implement these operations must adhere to the contract that they understand a *visit* call for each concrete class in the hierarchy. This is achieved through a common abstract base class (or an interface) *Visitor*. Consequently, the functionality of *scale* must be included in both

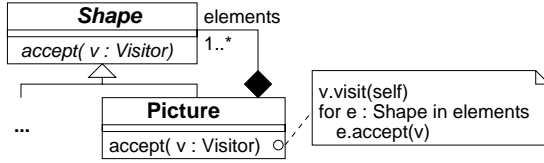


Figure 4. Traversal encoded in the objects

ScaleVisitor::visit(e:Line) and *ScaleVisitor::visit(e:Arrow)*. We will discuss this issue in Sect. 3.2.

Considering the dependencies between shapes and operations, we notice that shapes are now *decoupled* from subclasses of Visitor in the way that concrete Visitors (*ScaleVisitor*, *PersistVisitor*) are not known to concrete Shapes at compile-time. As an effect, we may change or add new operations on shapes without recompiling *Shape*, *Line*, *Arrow*, or *Picture*. Another side-effect regards attribute and method visibility: Since the *scale* methods now reside in *Scale*, they must have either access to the attributes in *Line* and *Picture*, or there must be some state-exposing interface in these classes. It is evident that applying VISITOR may break encapsulation, since at least the visitors need access to the object states.

A common variant of the VISITOR pattern is to leave the *traversal* through the object structure in the *accept* methods (Fig. 4). If all algorithms use the same way to iterate through the objects, redundant traversal code can be avoided. Perhaps even more important, the kind of composition can be left private, weakening the visitor’s dependency. On the other hand, future visitors are restricted to one kind of traversal. Thus, it seems to us, that this variant of the VISITOR pattern requires a lot of foresight, and should be applied carefully. Other design patterns may be used instead to avoid duplicated traversal code in the visitors (e.g. STRATEGY, ITERATOR [11]).

3. Problems Induced by VISITOR

Apart from problems that are intrinsic to the general idea behind VISITOR, like breaking encapsulation and introducing a level of indirection, there are several other problems that are solved in certain approaches. We address three of those problems with our DYNAMIC DISPATCHER.

3.1. Visitor is Intrusive

It is obvious that applying VISITOR to extend existing software by either extracting or creating new operations is a very intrusive procedure. Firstly, the domain classes must provide an *accept* method. Furthermore, for inherited methods explicit delegation code must be created due to the problem of having no implementation inheritance (see below). Therefore, VISITOR is definitely a heavy-weight pattern, which may not be applicable to existing software in

several cases. This problem is also discussed in [16], which presents a generic ‘Walkabout’ class to replace *accept*.

Another undesirable effect of VISITOR is the cyclic dependency relationship between classes and subclasses: superclasses know their subclasses, because Visitor knows all domain classes through the parameter types of its *visit* methods, and each domain class knows Visitor. [14] and [15] avoid cyclic dependency in ACYCLIC VISITOR and EXTRINSIC VISITOR.

3.2. No Implementation Inheritance

The VISITOR design pattern does not support implementation inheritance. In a design without a visitor (operations are defined in the classes they operate on), subclasses inherit the methods they do not override. In our example, *Arrow* inherits *scale* from *Line*. Nevertheless, *ScaleVisitor* needs to implement *scale(a:Arrow)* - which may call *scale(l:Line)*. Thus, implementation inheritance must be manually simulated when applying VISITOR in the general case.

Assuming we left out *visit(a:Arrow)* in the Visitor class, arrows would be handled as lines in each subclass of Visitor (i.e. in each extracted operation). As soon as one visitor must differentiate between lines and arrows, all visitors must do so. The reason is *again* the typical OO language’s uni-dispatch late-binding: because calls to *visit* are early bound with regard to the argument type, *visit* can not be specialized in a visitor. If our language supported covariant method overriding (or multimethods) like Dylan [2] and CLOS [9], this kind of specialization would be possible. Manually simulating implementation inheritance can become time-consuming, hard to maintain, and error prone: If one Visitor needs a special method for a certain domain class not yet present in the Visitor base class, all other visitors must be changed as well. Therefore, as discussed in [19], it is not possible to use VISITOR in frameworks/libraries, because the framework user cannot change the interface of the visitor base class, even if she is allowed to introduce new domain subclasses. In all cases, changes to the inheritance relationship between domain classes must be reflected in the manually coded delegation, with the danger of introducing hardly traceable errors.

One solution to overcome these problems is DEFAULT VISITOR [15], which introduces a common base class (*DefaultVisitor*) for visitors to inherit from. In *DefaultVisitor*, each *visit* method calls the next more general method, up to the most general argument type. Thus, individual visitors only need to override some *visit* methods. Still, DEFAULT VISITOR is not applicable to frameworks.

Another solution is to move the dispatching to the correct *visit* method into its own method (*dispatch*) in the visitor. This method checks the runtime type of its argument and calls the most specific *visit* method. This way, implementation inheritance is simulated, and additional Shape classes

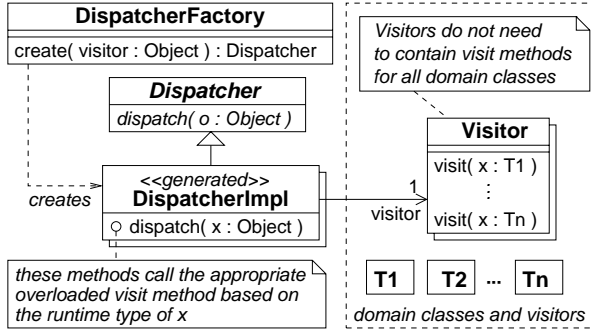


Figure 5. Structure of DYNAMIC DISPATCHER

are not intruded with *accept* methods. The major drawback of this approach is that we must manually maintain several *dispatch* methods (e.g. one for Scale and another for Persist). This solution is called EXTRINSIC VISITOR by [15].

4. Dynamic Dispatcher

Reconsidering how we motivated applying VISITOR, we recall that we tried to separate polymorphic operations from the domain objects they work on. VISITOR was motivated by the fact, that Java and most other common object-oriented programming languages only support uni-dispatch late-binding. If we had a language which supports at least double-dispatch method binding, it would not be necessary at all. Especially for Java, several approaches were discussed to introduce double-dispatch method binding (e.g. [10]). Most approaches we know require a modified compiler, or a modified virtual machine, they may not be adoptable for many applications. One exception is the Java Multi-Method Framework [18], which covers general multi-methods via reflection, but introduces a significant performance overhead to normal method invocation.

In the following, we present another solution DYNAMIC DISPATCHER, which specially targets the cases where VISITOR may be applied, and which does not require any changes to the compiler, virtual machine, or runtime. At its heart, we introduce a *Dispatcher* object, that dynamically chooses the most appropriate *visit* method. Thus, DYNAMIC DISPATCHER replaces the *accept* methods of the VISITOR pattern by an explicit dispatching object. This object is generated at runtime by passing a visitor object to a factory that dynamically derives the dispatcher object.

4.1. Structure

The general structure is depicted in Fig. 5. The classes *Dispatcher* and *DispatcherFactory* are part of the dispatching framework, while *Visitors* are arbitrary user defined classes that declare some *visit* methods. There is no abstract

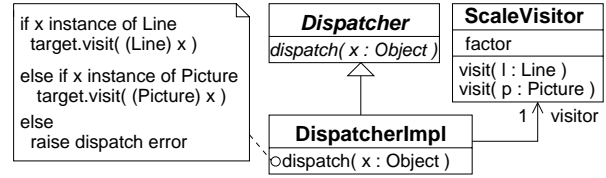


Figure 6. Dynamic dispatcher for Scale

base class *Visitor*, as in the original pattern, which requires a fixed set of *visit* methods to be defined.

A *DispatcherImpl* object is generated by *DispatcherFactory::create* and holds a reference to its visitor. The dispatching, i.e. calling the appropriate *visit* method, is done in the overridden *dispatch* method. Notice, that although not shown here, a *Visitor* may need a reference to the *Dispatcher* to (recursively) invoke its own *visit* methods, like in traversal operations. Although not strictly necessary, the *DispatcherImpl* class should be typically generated at runtime, as explained below.

4.2. Choosing Appropriate Visit

What does choosing the appropriate *visit* method for an argument *x* of *dispatch* mean? Basically, if there is more than one *visit* method with an argument type to which *x* is assignable, the one with the most specific argument type should be called. In general, in programming languages with subtyping (see [6, 1]), there may be no most specific type. This is the case if none of the assignable argument types is a subtype of all others. If there is no most specific type for *x*, then there is no most appropriate *visit* method. We regard this situation as an ambiguity error.

For Java and C#, we can avoid this ambiguity by restricting the allowed argument types in the *visit* methods to class types. Because Java and C# prohibit multiple inheritance between classes, this restriction ensures that if there is at least one compatible *visit* method for *x*, there is always a most specific method. Less restrictive solutions exist (e.g., [5],[18]), but they are more complicated. However, if our intention was to move operations out of a class hierarchy, we do not impose any restrictions, because the only place a method can be defined in Java is a class.

Let us explain a dispatcher generated for a *ScaleVisitor* that works on our shape-hierarchy, as depicted in Fig. 6. The visitor consists of *visit* methods for *Line* and *Picture*. The *dispatch* method sequentially checks the parameter against the types of the *visit* methods, and calls the appropriate one. Notice, that we gained ‘implementation inheritance’. In the previous section, we have defined *Arrow* as a subclass of *Line*, so that the expression *x instance of Line* yields true for an instance *x* of *Arrow*. Hence, passing an *Arrow* object to *dispatch* results in the execution of

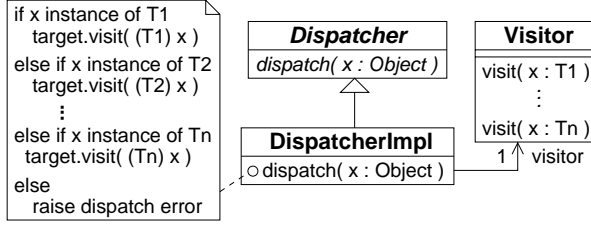


Figure 7. Simple dispatch algorithm

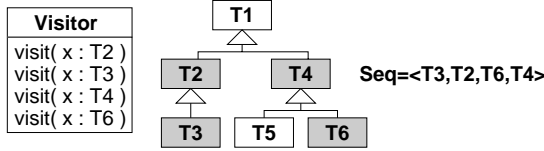


Figure 8. Deriving a sequence for dispatch

visit(l:Line). This corresponds with our intention that scaling arrows is inherited from scaling lines.

In contrast, persisting shapes differentiates between arrows and lines, expressed by a separate *visit(a:Arrow)* method in *PersistVisitor*. Therefore, the corresponding dispatcher must also check against class *Arrow*. Actually, this check must happen before the one against *Line*, otherwise *visit(l:Line)* would be called unintendedly.

In general, the *dispatch* method can be implemented as a sequence of *if*-statements. The inheritance tree between classes (without multiple inheritance) can be transformed into a sequence T_1, \dots, T_n , so that the *dispatch* method looks like Fig. 7. For T_1, \dots, T_n , $(T_i \text{ subtype of } T_j) \Rightarrow i < j$ must hold to ensure that more specific types are checked first.

Figure 8 illustrates how to derive such a sequence from a set of types. The grey-shaded boxes in the class diagram are the classes occurring as argument types of the *visit* methods. The depicted sequence of types is one of the four valid sequences for the simple dispatch algorithm used in Fig. 7.

One may argue that the implementation as a sequence of *if*-statements is against the spirit of object-orientation. After all, the removal of conditional statements is one major advantage of OO. Yet, *dispatch* enables us to express visitor objects in a far more natural way. We gain simplicity in many other places by violating the OO paradigm in one place. Furthermore, as shown later, the dispatcher implementation can easily be generated on demand at runtime, removing the drawback of manually coded *if*-statements.

Other implementation techniques of *dispatch* are possible. For deep class hierarchies, it may be more efficient to reorder the *if*-statements to perform a tree search along the class hierarchy to find the appropriate *visit* method. More complex, already found method resolutions may be

cached (e.g. in a hash table), or precomputed (e.g. as in [17]). Unfortunately, the presence of dynamic linking and multi-threading requires synchronization. We experienced that the simple linear search algorithm is sufficient in many cases (see Sect. 5).

4.3. Consequences and Requirements

In Sect. 3 we discussed three major problems of the VISITOR pattern: intrusiveness, cyclic dependencies, and the lack of implementation inheritance. DYNAMIC DISPATCHER addresses these problems.

Our variant allows to define new functionality over domain classes in the same way VISITOR does. In contrast to VISITOR, it does not require changes to the domain classes. Thus, it is not intrusive and can be applied to extend existing frameworks and libraries. As a consequence of the fact that domain classes do not need an *accept* method in our approach, DYNAMIC DISPATCHER does not introduce cyclic dependencies between domain classes.

Finally, we simulate covariant overriding of *visit* methods. Therefore, DYNAMIC DISPATCHER allows to simulate implementation inheritance in visitor operations. This means that developers who add visitor operations to domain classes can *specialize* them for individual domain classes in the same natural way they do when they simply override methods in the domain class hierarchy. Consequently, DYNAMIC DISPATCHER based implementations are better maintainable and more robust against future changes to domain classes as well as changes to individual visitors. Also, it allows developers to express functionality clearer and more concise than VISITOR, because no dispatching code clutters the visitor class.

In languages with dynamic linking of types, the dispatching algorithm cannot be derived at compile time. At least, we need some kind of *runtime type reflection* mechanism to analyze the method signatures of a certain visitor class. That is, which *visit* methods are available and which are more special than others. We also need a way to determine if an object is an instance of a certain type to find the most appropriate method.

The actual dispatching of an invocation can be implemented via reflection, if dynamic method invocation is supported. Alternatively, dispatching code can be generated on the fly. We present a small framework in Sect. 5, which we used to evaluate both dispatching variants for Java.

There are two weak points in comparison to VISITOR: The first is, DYNAMIC DISPATCHER may decrease performance due to the additional dispatch code to be performed. We will discuss some performance measurements we conducted in Sect. 6. The second is, DYNAMIC DISPATCHER performs type checking at runtime. If no appropriate *visit* method is found, some kind of dispatch error must be raised. Therefore, in DYNAMIC DISPATCHER type errors can occur

at runtime, whereas they are detected by the compiler in the VISITOR pattern.

5. Implementation

For evaluation purposes, we have developed a small framework that realizes DYNAMIC DISPATCHER in Java. In the following, we give a brief overview of the implementation and discuss some performance results.

We have realized three different dispatcher factories. One can choose each of these to create an instance of Dispatcher for a particular object that contains *visit* methods.

Our first solution (SCDispFactory) is straightforward: *SCDispFactory::create* analyzes the given visitor object for *visit* methods. The argument types are extracted, and ordered as depicted in Fig. 8. Then, source code for a Java class that implements *dispatch* as in Fig. 6 is written to a temporary text file and compiled at runtime by the Java Compiler interface. The resulting class is loaded through a custom class loader and finally an instance of it is and returned.

The second solution (ReflectiveDispFactory) reuses a generic class that is initialized with the type sequence as described above. Whenever *dispatch* is called on this object it iterates over the sequence until the first assignable type is found and invokes the corresponding method by reflection.

Both solutions suffer from several problems which we will discuss later on. Our last dispatcher factory (BCDispFactory) avoids these problems. It works similar to the source code generator, but instead of compiling the dispatcher class from a temporary text file it directly defines the class in byte code. The byte code is generated using the free Bytecode Engineering Library (BCEL) [8] which provides helper classes for writing Java byte code.

6. Performance Analysis

To estimate the impact of our DYNAMIC DISPATCHER implementation, we have conducted two performance suites where we compared the three aforementioned dispatcher factories with the original visitor, and of course with the initial, object-oriented form. We will discuss the results at the end of this section. All tests were executed on a single user, 1.2 GHz Pentium III machine running Windows XP and J2SDK, Version 1.4.2.

6.1. Test Suites and Results

The first suite ('raw') is meant to estimate the cost of a single method dispatch. Because our three dispatcher implementations all perform a linear search to find the appropriate method, we parameterized this suite with the number of classes n . For a single run we create n subclasses C_i of a Base class, each overriding the Base methods f and *accept*. For VISITOR and DISPATCHER, we measure the time

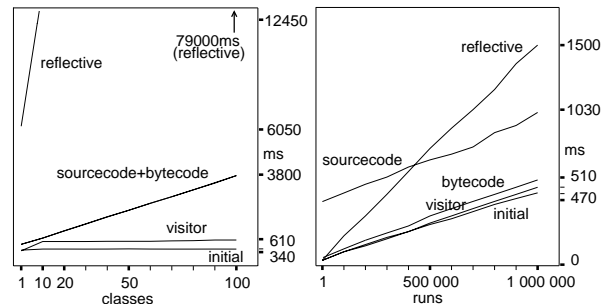


Figure 9. Performance comparison

to invoke *visit* through *accept* resp. *dispatch* with randomly chosen C_i instances as arguments. For the initial form, we measure the time of a simple call of f . To eliminate the effects of test setup, we perform a large number of dispatches (ten millions) and show the total time. In our simplistic linear search approach it makes no difference whether the class hierarchy is flat or deep.

The left-hand side in Fig. 9 shows the results of this suite: the initial form, which is a simple method invocation of f , takes a constant amount of time. As estimated, the execution time of source and byte code generated dispatchers is nearly equal, as they only differ in the way they are generated. They grow in a linear way with the number of classes n . For $n = 1$ they take twice as much time to execute as the initial form, with $n = 100$ they take ten times the amount. The reflective dispatcher is much slower: its ratio grows from 18 at $n = 1$ to 230 at $n = 100$. The visitor takes always about twice the amount of the initial form, except for $n = 1$, where they are equal. We wondered about this point and found out, that this break out vanishes, when Java's just-in-time compiler is deactivated.

While the first suite measures the raw cost of method invocations, the second one ('scale') measures the overall effect of using a dispatcher in a complete algorithm. We have implemented the Scale algorithm in Fig. 1 and 2 in all forms (initial, visitor, byte code, source code, and reflective dispatcher). Then we scaled a simple picture, which consists of arrows and lines multiple times, and measured the overall execution time. In contrast to the first suite, we included the time required to construct the dispatcher and visitor objects.

The result is shown in the right-hand side of Fig. 9. As expected, all graphs increase in a linear way. The reflective dispatcher is by far the slowest while the others perform nearly equal. After one million runs, the difference between byte code dispatcher and the initial form is less than ten percent. The source code generated dispatcher increases by the same degree as the byte code generated dispatcher, but takes a quite large amount of time to construct the dispatcher class.

6.2. Performance Consequences

Our test suites are not exhaustive, but they give some hints about how expensive our approach is. Method invocation through the byte code generated dispatcher is slower than normal method invocation and invocation through an *accept* method. However, the relative overhead decreases significantly if some (even little) code is executed within the invoked method. To provide a number, the overhead in our graphics example is about ten percent. We expect that this overhead is negligible in many real world scenarios.

If only few calls are to be performed, the reflective dispatcher may also be sufficient. Its advantage is that it can be implemented as a single generic reusable class.

7. Conclusion and Future Work

We presented a variant of the VISITOR pattern that is more suitable for framework designs and more natural for software developers in certain situations. It can be applied to extend software without affecting existing code. We understand it as another tool that is especially useful in agile software development.

We demonstrated how DYNAMIC DISPATCHER can be implemented as a small framework in Java. We could have done this in C# and the .NET framework as well, since .NET also provides the required reflection capabilities. For C++, at least the sketched generated source code dispatcher can be implemented. Our evaluation showed that the performance tradeoff implied by the implementation should be acceptable in many real world situations.

One of the drawbacks of DYNAMIC DISPATCHER is that type errors can occur at runtime. We are currently working on an extension to achieve more static type checking. Basically, we are going to allow to specify a user defined base type for the dispatcher object.

Apart from these technical aspects we are going to further evaluate the applicability of DYNAMIC DISPATCHER for a larger project in which the aforementioned problems of VISITOR arise. We are going to realize the project with both variants. We expect that the DYNAMIC DISPATCHER solution will take less time to be completed and results in a simpler overall design.

References

- [1] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer, New York, 1996.
- [2] Apple Computer, Eastern Research and Technology. *Dylan: an object-oriented dynamic language*, 1992.
- [3] K. Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.
- [4] A. Beugnard. OO languages late-binding signature. Ninth International Workshop on Foundations of Object-Oriented Languages, 2002.
- [5] J. Boyland and G. Castagna. Parasitic methods: Implementation of multi-methods for Java. In *Conference Proceedings of OOPSLA '97, Atlanta*, volume 32(10) of *ACM SIGPLAN Notices*, pages 66–76, New York, NY, 1997. ACM.
- [6] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.
- [7] G. Castagna. Covariance and contravariance: Conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, 1995.
- [8] M. Dahm. Byte code engineering with the BCEL API. In C. H. Cap, editor, *Java Informationstage '99, Informatik aktuell*, pages 267–277, 1999.
- [9] L. G. DeMichiel and R. P. Gabriel. The Common Lisp Object System: An overview. In J. Bezivin et al., editors, *ECOOP '87, European Conference on Object-Oriented Programming, Paris, France*, pages 151–170. Springer, New York, NY, 1987. LNCS, Volume 276.
- [10] C. Dutchyn, P. Lu, D. Szafron, S. Bromling, and W. Holst. Multi-Dispatch in the java virtual machine: Design and implementation. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS-01)*, pages 77–92, 2001.
- [11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1997.
- [12] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall, 2001.
- [13] R. C. Martin. Acyclic visitor. In R. C. Martin, D. Riehle, and F. Buschmann, editors, *Pattern Languages of Program Design 3*, pages 93–104. Addison-Wesley Publishing Co., Reading, MA, 1998.
- [14] R. C. Martin. *Agile Software Development. Principles, Patterns, and Practices*. Pearson Education, 2002.
- [15] M. E. Nordberg. Default and extrinsic visitor. In R. C. Martin, D. Riehle, and F. Buschmann, editors, *Pattern Languages of Program Design 3*, pages 105–123. Addison-Wesley Publishing Co., Reading, MA, 1998.
- [16] J. Palsberg and C. B. Jay. The essence of the visitor pattern. In *Proc. 22nd IEEE Int. Computer Software and Applications Conf., COMPSAC*, pages 9–15, 1998.
- [17] C. Pang, W. Holst, Y. Leontiev, and D. Szafron. Multi-method dispatch using multiple row displacement. *LNCS*, 1628:304–329, 1999.
- [18] Remi, F. Etienne, and D. Gilles. Java multi-method framework. *International Conference on Technology of Object-Oriented Languages and Systems (TOOLS'00)*, 2000.
- [19] J. Vlissides. Visitor in frameworks. *C++ Report*, 11(10):40–46, 1999.

Document Clustering with Adaptive Term Weighting and Feature Reduction Capabilities

T.W. Fox and B.J. Fox
Intelligent Engines
tfox@bm.net

Abstract. Document clustering is a powerful data mining technique for topic discovery that can be used to organize a document corpus into groups of similar documents. This paper introduces adaptive term weighting to simultaneously discover the document clusters and the discriminatory terms. The use of adaptive term weights can increase the total F-measure of the final cluster hierarchy by as much as 32% over previous term weighting methods. A document feature reduction method is introduced to remove non-discriminatory terms from the document feature space without degrading the final cluster hierarchy. This method can be used to reduce the storage requirement of the clusters by as much as 79%.

1. Introduction

The World Wide Web (WWW) continues to grow at an amazing speed. Consequently, hand-built directories that group similar documents are becoming increasingly difficult to maintain for the internet and for other large repositories of documents. A more practical solution is to discover topic hierarchies using an unsupervised document clustering method.

Document clustering is a common data mining technique for topic discovery. A document cluster is a homogeneous group of documents that are more strongly associated with each other than documents in different groups [1]. Document clustering provides convenience for exploring the content of a document corpus because document clustering organizes the document corpus into groups of similar documents. Document clustering can be used to organize search results into groups based on document similarity, which helps the user find relevant documents more quickly [2]. Document clustering is also useful for Information Retrieval (IR). In Clustered Based Retrieval (CBR), a query vector is compared to each cluster centroid [3]. Documents contained in the most relevant clusters are retrieved [3].

K-means is widely used in large scale document clustering because of its speed and simplicity [1,2]. K-means begins by initializing K cluster centroids to random

locations in the document feature space. Each data point is assigned to the nearest (most similar) cluster centroid. Each cluster centroid is then re-calculated. The k -th cluster centroid r_k is computed as the vector mean of the data points that are assigned to the k -th cluster. This process of assigning data points and re-calculating cluster centroids is repeated until the cluster centroids converge.

Variants to K-means have been presented in the literature. For example, Scatter/Gather [4] uses K-means to refine the results from hierarchical clustering. In this method, hierarchical clustering [1] is used to determine K. Unfortunately, the hierarchical clustering portion of Scatter/Gather can be slow [1]. Consequently, Scatter/Gather is most suited for smaller data sets. Bisecting K-means [5] uses K-means to partition the dataset into two clusters. The largest cluster is partitioned into two clusters. This process repeats until K clusters have been discovered. Unfortunately bisecting K-means can be slow for large data sets because K-means is used in each iteration to re-partition the data.

Despite its widespread use, K-means has several limitations when applied to document clustering. K-means does not reduce the dimensionality of the feature space. This is of concern mainly for document clustering because of the large number of features (terms) required to represent documents [1] (typically above 10,000). Consequently, storage requirements for the final clusters can be extremely high. The dimensionality of the document feature space can be reduced prior to clustering, as is suggested in [2,6]. Dhillon, Kogan and Nicholas present a term variance quality measure in [6], which ranks each unique term in the corpus. They suggest that terms with low term variance quality values can be removed from the feature space. Similarly, the method presented in [2] reduces the document feature space to the top twenty terms as ranked using Term Frequency-Inverse Document Frequency (TF-IDF) weighting. Unfortunately both of these methods may inadvertently remove terms that are essential to obtain the optimal cluster hierarchy, as the results presented in [2] suggest. In contrast, this paper presents a new document clustering method to

preserve terms that are essential to the cluster hierarchy (discriminatory terms) while removing terms that are not.

Adaptive term weighting is introduced in this paper to identify discriminatory terms. During each iteration, the adaptive term weights are chosen to optimize the within cluster metric. Discriminatory terms are given high values, and non-discriminatory terms are given low values. An algorithm is presented to selectively remove terms with low weights from the document feature space without affecting the final cluster hierarchy. It is shown that it is possible to reduce the storage requirement by as much as 79% without degrading the final cluster hierarchy. The use of adaptive term weights also increases the total F-measure of the final cluster hierarchy by as much as 32% over previous weighting methods.

2. Description of the Proposed Document Clustering Method

The proposed document clustering algorithm is based on K-means. However, new term weighting and feature reduction methods have been incorporated. Each unique term in the document corpus is given a weight, called an adaptive term weight, whose value changes with each iteration. The adaptive term weights can take on values between zero and one. A value close to one signifies that the term is discriminatory, and a value close to zero signifies that the term is non-discriminatory.

The proposed document clustering algorithm begins by extracting features from the document corpus to form vectors. A vector is created for each document. Section 2.1 describes the application of the vector space model to represent documents as vectors. The algorithm then initializes K cluster centroids to random locations in the feature space. The adaptive term weights are determined, as described in Section 2.3. Each document is assigned to the most similar cluster centroid. Section 2.2 presents the weighted cosine similarity measure, which is used to quantify the similarity between documents. The cluster centroids are re-calculated. The k -th cluster centroid r_k is computed as the vector mean of the data points that are assigned to k -th cluster. This process of updating the adaptive term weights, assigning data points and re-calculating cluster centroids is repeated until the cluster centroids converge. The last step is feature reduction. The final values of the adaptive term weights are used to selectively remove features without degrading the document clusters. Section 2.4 presents the feature reduction method.

2.1. Document Representation

The document clustering method proposed in this paper uses the vector space model, which was introduced in [7,8]. In the vector space model, each document is

represented as a vector, and each document vector has p elements, where p is the total number of unique terms in the corpus. A document vector is as follows:

$$v_j = [t_{1j}, t_{2j}, \dots, t_{pj}]^T \quad (1)$$

where v_j is the j -th document vector. The value t_{ij} quantifies the occurrence of i -th term in the j -th document. The principle advantage of using the vector space model is that vector arithmetic operations can be used to quantify similarity between documents.

Term Frequency (TF) and Term Frequency-Inverse Document Frequency (TF-IDF) weighting are commonly used in the vector space model [1]. In TF weighting, t_{ij}^{TF} represents the number of times the i -th term occurs in the j -th document [1]. TF-IDF weighting is defined as [1]:

$$t_{ij}^{TF-IDF} = TF_{ij} \cdot \log(N / TF_{ij}) \quad (2)$$

where N is the total number of documents in the corpus and TF_{ij} is the term frequency of the i -th term in the j -th document [1].

TF and TF-IDF weighting do not consistently produce high values for discriminatory terms and low values for non-discriminatory terms [6]. Instead, this paper introduces adaptive term weighting to consistently assign high values for discriminatory terms and low values for non-discriminatory terms.

The adaptive term weight for the i -th term in the j -th document is expressed as

$$t_{ij}^{adaptive} = w_{ki} \cdot TF_{ij} \quad (3)$$

where w_{ki} is the adaptive term weight of the i -th term for documents that belong to the k -th cluster ($k = 1, 2, \dots, K$). Each document cluster is given its own set of adaptive term weights because the distribution of words can vary drastically from one cluster to another. Documents that belong to the k -th cluster use the k -th set of adaptive term weights to calculate t_{ij} .

The value of w_{ki} is resolved during clustering. As the clusters form, the adaptive term weights increase in value for discriminatory terms and decrease in value for non-discriminatory terms. In contrast, TF-IDF weighting is constant throughout clustering. TF-IDF weighting cannot be used to consistently detect discriminatory terms because TF-IDF weighting is computed with no regard to the cluster structure. Section 2.3 presents a method to update w_{ki} at each iteration of the clustering process.

The chief disadvantage of the vector space model is its high dimensionality. It is not uncommon for the vectors to have more than 10,000 elements. To reduce dimensionality and to improve cluster quality, word

stemming and stop words are used. Stop words are non-content bearing words, such as “and” and “or”, that can be removed without affecting clustering results. Word stemming involves the removal of any attached suffixes and prefixes from the word to yield the word stem. Word stemming therefore reduces the number of distinct terms, which reduces storage and processing time. The method proposed in this paper uses Porter’s algorithm [9] to stem all of the words in the corpus.

2.2. Quantifying Similarity Between Documents

Various vector arithmetic operations can be used to calculate the similarity between documents, such as Euclidean distance or cosine similarity [1,6]. Cosine similarity between two vectors, v_1 and v_2 , with p elements is defined as [1]

$$S(v_1, v_2) = \frac{\sum_{i=1}^p v_{1i} \cdot v_{2i}}{\sqrt{\sum_{i=1}^p v_{1i}^2} \sqrt{\sum_{i=1}^p v_{2i}^2}} \quad (4)$$

The cosine similarity measure is the cosine of the angle between the two vectors [1]. The values of this similarity measure are constrained between zero and one because each vector element must be greater or equal to zero for the document clustering problem [1]. Very similar documents have cosine values close to one, and very dissimilar documents have cosine values close to zero [1]. The cosine similarity measure is better suited for document clustering than Euclidean distance because the cosine similarity measure offers greater numerical separation between dissimilar documents [1].

We now modify the cosine similarity measure to accommodate adaptive term weights. The proposed K-means algorithm compares a document vector to a cluster centroid. Each cluster is associated with an adaptive term weight vector. Documents that are similar to the cluster centroid must contain cluster discriminatory terms, which have high adaptive term weights. The other non-discriminatory terms should have reduced effect on the similarity measure. To this end, we define the weighted cosine similarity measure as:

$$S_w(v_j, r_k) = \frac{\sum_{i=1}^p w_{ki} \cdot TF_{ij} \cdot r_{ki}}{\sqrt{\sum_{i=1}^p TF_{ij}^2} \sqrt{\sum_{i=1}^p r_{ki}^2}} \quad (5)$$

where w_{ki} is the i -th term weight of the k -th cluster, TF_{ij} is the frequency of the i -th term in the j -th document, r_k is the k -th cluster centroid vector and v_j is the j -th document

vector. Each term weight is permitted values between $0 \leq w_{ki} \leq 1$. During clustering, discriminatory terms are given weights approaching one and non-discriminatory terms are given weights approaching zero. In this way, discriminatory terms numerically dominate the weighted cosine similarity measure. As with the cosine similarity measure, very similar documents have weighted cosine values approaching one, and very dissimilar documents have cosine values close to zero.

The within clustering (distortion) metric for the document clustering problem is defined as [1]

$$wc(C) = \sum_{k=1}^K \sum_{v \in C_k} S(v, r_k) \quad (6)$$

where C represents the set of K clusters, v corresponds to a document vector and r_k is the k -th cluster centroid. Higher values for $wc(C)$ are desirable because they indicate that data points within the clusters are very similar to their assigned cluster centroids. The weighted with-in clustering metric is defined as:

$$wc_w(C, w) = \sum_{k=1}^K \sum_{v \in C_k} S_w(v, r_k) \quad (7)$$

2.3. Updating the Adaptive Term Weights

The adaptive terms weights are chosen to reflect the relevance of each term to a given cluster and to increase $wc(C)$. The term weights can be chosen by solving the following constrained optimization problem:

$$\text{Maximize } \gamma(w) \quad (8)$$

$$\text{Subject to } 0 \leq w_{ki} \leq 1$$

$$\text{for } i = 1, 2, \dots, p \text{ and } k = 1, 2, \dots, K \quad (9)$$

where w is a matrix composed of the individual adaptive term weight vectors

$$w = [w_1, w_2, w_3, \dots, w_K] \quad (10)$$

and $\gamma(w)$ is calculated using the Algorithm (1).

Calculating $\gamma(w)$ does not alter the document assignments or cluster centroids. This constrained optimization problem is very difficult to solve. It is not uncommon for w_k to contain over 10,000 parameters. Also the objective function is discontinuous, which precludes the use of derivative based optimization methods. Derivative-free optimization methods, such as Simulated Annealing [10] and the downhill simplex method [11] can be modified to solve this optimization

problem. However, the computational time required to solve this optimization problem (with over 10,000 variables) is extremely high. Instead this section introduces a faster optimization algorithm that can be used to provide approximate solutions to this constrained optimization problem.

1. Save the current document cluster assignments and cluster centroids.
2. Assign documents to the most similar cluster centroid using S_w .
3. Recalculate the cluster centroids.
4. Set $\gamma(w) = wc(C)$.
5. Restore the original document cluster assignments and cluster centroids.

Algorithm 1. Calculating the value of $\gamma(w)$.

The proposed optimization method first uses a heuristic to generate an initial trial solution and then uses a random local search to improve upon this initial trial solution. The heuristic is as follows:

1. Calculate the normalized contribution of the i -th term in the k -th cluster to $wc(C)$ using the following relation:

$$\delta_{ki} = \frac{1}{\sum_{v_j \in C_k} S(v, r_k)} \sum_{v_j \in C_k} \frac{r_{ki} \cdot w_{ki} \cdot TF_{ij}}{|r_k| |TF_j|} \quad (11)$$

where r_{ki} is the i -th element in the k -th centroid, v_{ji} is the i -th element in the j -th document vector, and TF_j is the term frequency vector used in v_j .

2. Set the i -th element of the k -th term weight as follows: $w_{ki} = \delta_{ki}^e$
3. The exponent e is chosen to solve the following one-dimensional optimization problem:

$$\text{Maximize } \gamma(\delta^e) \quad (12)$$

$$\text{Subject to } 0 \leq e \leq 1 \quad (13)$$

Algorithm 2. Generation of an initial trial solution.

The above heuristic yields an initial trial solution to the optimization problem defined by equations (8) and (9). Each term weight equals the normalized contribution of each term to $wc(C)$ raised to a power e . Power e is chosen to maximize $\gamma(\delta^e)$ such that $0 \leq e \leq 1$.

The normalized contribution of the i -th term in the k -th cluster to $wc(C)$ provides a guess of the solution to the

problem defined by equations (8) and (9). The one-dimensional optimization problem defined by equations (12) and (13) refines this guess.

This initial trial solution is feasible but it may not correspond to a constrained local maximum. A random local search is used to improve upon this initial trial solution. In this random local search, x_c is the current position vector in the parameter space and x_t is a trial solution vector. The algorithm is as follows:

1. Obtain an initial trial solution using Algorithm 2.
2. Repeat the following N_{tot} times:
 - a. Set $x_t = x_c + \Delta$, where Δ is a random vector with each element drawn from interval $[-\varepsilon, \varepsilon]$ such that $0 \leq x_{ti} \leq 1$ for $i = 1, 2, \dots, p$. The constant ε is a small number.
 - b. If $\gamma(x_t) > \gamma(x_c)$ then set $x_c = x_t$.
 - c. Terminate if x_c has converged.

Algorithm 3. A random local search used to calculate the adaptive term weights.

Because of the high dimensionality of this optimization problem, significant computational time must be expended to converge on a constrained local minimum. Setting N_{tot} to a finite value limits the computational time. The resulting solution will be feasible but it may not correspond to a constrained local minimum.

1. Set $w_0 = w$, $\varepsilon = \text{small number}$, $\tau = \varepsilon$, $\theta = 0.01\%$.
2. Repeat the following until all changes in w are rejected:
 - a. Set all elements of w to zero that are less than or equal to τ .
 - b. If $\frac{|wc_w(C, w_0) - wc_w(C, w)|}{wc_w(C, w_0)} < \frac{\theta}{100}$ then accept the change in w and increase τ . Otherwise reject the change.

Algorithm 4. Document feature reduction.

2.4. Feature Reduction

The final cluster hierarchy must be stored for future use. Because of the high dimensionality of the vector space model, the storage requirement for the final cluster

hierarchy is very high. This section proposes a method to reduce the storage requirement by as much as 79%.

The adaptive term weights are fixed after the clustering process has completed. Each term has a weight value between zero and one. Terms that have weights close to one are considered discriminatory and contribute significantly to $w_{C_w}(C)$. Terms that have zero weight do not require storage in the database because those terms do not contribute at all to $w_{C_w}(C)$. Terms that have near zero weight contribute very little to $w_{C_w}(C)$. Many of these terms can be removed from the feature space without affecting the cluster hierarchy.

Algorithm 4 can be used to select which terms to remove from the feature space at the expense of a slight change to $w_{C_w}(C)$, which is controlled by θ . Increasing the value of θ reduces the feature space, but at the expense of altering the cluster configuration (if K-means is re-started). A small value of θ should be used to ensure that the cluster configuration does not change if K-means is restarted. Experimentation revealed that setting $\theta = 0.01\%$ is sufficiently low to preserve the cluster configurations for the test documents used in this paper.

The example presented in the next section demonstrates that it is possible to reduce the feature space by one-fifth. This reduction in feature space does not affect the final cluster hierarchy because restarting the clustering algorithm after feature reduction does not alter any document assignments.

The feature reduction method proposed in this section differs significantly from previous feature reduction methods. The feature reduction methods presented in [2,6] ranks each term in the feature space prior to clustering. The top ranked terms are retained while the bottom ranked terms are removed. Clustering occurs after the feature space has been reduced. Unfortunately this approach may inadvertently remove terms that are essential to obtain the optimal cluster hierarchy, as the results presented in [2] suggest. In contrast, the method proposed in this paper discovers the cluster hierarchy and discriminatory terms simultaneously. This information is used to reduce the feature space without affecting the cluster hierarchy.

3. Experimental Verification

This section demonstrates that the proposed method can be used to significantly reduce the dimensionality of the feature space without degrading any of the clusters. This section also demonstrates that the quality of the clusters obtained using adaptive terms weights is significantly higher than using either term frequency or TF-IDF weighting.

A pool of multi-page documents has been collected and categorized by human judges. Our cluster evaluation method compares how closely each cluster generated by

the clustering system matches the set of categories previously assigned to the documents by human judges.

The “total F-measure” discussed in [2] is used to score cluster quality. To calculate the total F-measure, the following data must be available:

- N_1 = number of documents judged to be of topic T in cluster X [2].
- N_2 = number of documents in cluster X [2].
- N_3 = number of documents judged to be of topic T in the corpus [2].

With this data, the precision and recall for cluster X can be calculated using the following relations:

$$P(X, T) = \frac{N_1}{N_2} \quad (14)$$

and

$$R(X, T) = \frac{N_1}{N_3} \quad (15)$$

where R measures recall and P measures precision. The F-measure for cluster X is [2]

$$F(T) = \frac{2PR}{P + R} \quad (16)$$

The total F-measure is defined as [2]

$$F_{tot}(T) = \frac{\sum_{T \in M} T_{tot} F(T)}{\sum_{T \in M} T_{tot}} \quad (17)$$

where M is the set of topics, T_{tot} is the total number documents of topic T.

Three test document data sets have been used to evaluate performance. The first data set, Corpus 1, is composed of 255 documents from the proceedings of the 1983 to 1985 Principles of Database Systems (PODS), the proceedings of 1997 ACM Special Interest Group on Management of Data (SIGMOD), and the proceedings of the 1975, 1980 and 1981 International Conference on Very Large Databases (VLDB). The second dataset, Corpus 2, is composed of 61 documents drawn from VLDB 1975, SIGMOD 1997, and from the proceedings of the 2000, 2001, 2002 IEEE International Symposium on Circuits and Systems (ISCAS). Documents from this data set can be divided into four categories: digital filters, power electronics, database performance and relational theory. The last data set, Corpus 3, is the same as Corpus

2 except additional papers dealing with Orthogonal Frequency Division Multiplexing (OFDM) have been added to the aforementioned documents. Corpus 3 contains 76 documents.

Table 1 shows the feature reduction results and Table 2 shows total F-measure results for the proposed method. Feature space storage is calculated as the total number of bytes required to store the final cluster hierarchy. Tables 1 and 2 also show results for K-means with term frequency and TF-IDF weighting.

Table 1. Feature Reduction Results

Term Weighting	Cluster Storage for Corpus 1, K=2 (bytes)	Cluster Storage for Corpus 2, K=4 (bytes)	Cluster Storage for Corpus 3, K=5 (bytes)
Adaptive	3151576	844016	962828
TF-IDF	14725208	3147800	3783212
TF	14725208	3147800	3783212

Table 2. F_{tot} Results

Term Weighting	F_{tot} for Corpus 1, K=2	F_{tot} for Corpus 2, K=4	F_{tot} for Corpus 3, K=5
Adaptive	0.86	0.89	0.90
TF-IDF	0.76	0.72	0.74
TF	0.7	0.66	0.68

The storage requirement for clusters generated by the proposed method is significantly lower than the storage requirement for clusters generated by K-means using either term frequency or TF-IDF weighting. For Corpus 1, the storage requirement for clusters generated by the proposed method is one-fifth that of clusters generated by TF and TF-IDF term weighting.

The feature reduction method proposed in this paper discovers the cluster hierarchy and discriminatory terms simultaneously. This information is used to remove non-discriminatory terms that do not affect the final clusters. Reducing the feature space in this way does not affect the final clusters because restarting the algorithm after feature reduction does not alter any document assignments.

Table 2 demonstrates that adaptive term weighting significantly increases the total F-measure. For Corpus 3, the total F-measure for adaptive term weighting is 0.22 and 0.16 higher than for TF and TF-IDF weighting respectively, which corresponds to a 32% and 21%

increase respectively. For each iteration, the adaptive term weights are chosen to optimize $w_c(C)$. Consequently, adaptive term weighting biases the search to favour cluster configurations that have high $w_c(C)$ values. In contrast, TF and TF-IDF weighting is static throughout clustering. Static weighting does not consistently force the search into high $w_c(C)$ configurations.

4. Conclusion

Adaptive term weighting has been introduced to discover discriminatory terms during clustering. Adaptive term weighting biases K-means to favour cluster configurations with high $w_c(C)$ values. This paper has also presented a feature reduction method to selectively remove non-discriminatory from the feature space without degrading the clusters. It is possible to reduce the storage requirement by as much as 79%.

References

- [1] D. Hand, H. Mannila, and P. Smyth, *Principles of data mining*, MIT Press, 2001.
- [2] B. Larsen and C. Aone, "Fast and effective text mining using linear-time document clustering", Proc. KDD, San Diego, 1999, pp. 16-22.
- [3] F. Can and E.A. Ozkaran, "Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases", ACM Trans. Database Systems, Vol. 15, No. 4, Dec. 1990, pp. 483-517.
- [4] D.R. Cutting, D.R. Karger, J.O. Pedersen, and L.W. Tukey, "Scatter / Gather: A Cluster-based Approach to Browsing Large Document Collection", Proc. ACM SIGIR, 1992, pp. 318-329.
- [5] M. Steinbach, G. Karypis, and V. Kumar, "A Comparison of Document Clustering Techniques", Proc. TextMining Workshop, KDD, 2000.
- [6] M.W. Berry. *Survey of text mining: clustering, classification and retrieval*, Springer, New York, 2004.
- [7] G. Salton, *The SMART Retrieval System*. Prentice-Hall, Englewood Cliffs, NJ, 1971.
- [8] G. Salton and M.J. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.
- [9] M.F. Porter, "An algorithm for suffix stripping", Program, Vol. 14, No. 3, 1980, pp. 130-137.
- [10] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, "Optimization by Simulated Annealing", Science, Vol. 220, No. 4598, 1983, pp. 671-680.
- [11] J.A. Nelder and R. Mead, A simplex method for function minimization", Computer Journal, Vol. 7, 1965, pp. 308-313.

Effort Estimation for Knowledge-based Configuration Systems

A. Felfernig

Business Informatics and Application Systems, University Klagenfurt

email: felfernig@ifit.uni-klu.ac.at

Abstract

Knowledge-based configuration is a successful application of Artificial Intelligence techniques in industrial environments. The increasing size and complexity of configuration problems, more expressive and model-based knowledge representation formalisms led to the implementation of large configuration applications. In this context an effective support of cost estimation for configuration software development is a crucial factor for project management. Based on our experiences in implementing configurators in various industrial environments we discuss aspects of the application of Function Point Analysis (a widely applied cost estimation approach in Software Engineering) in the context of knowledge-based configuration projects.

1. Introduction

Knowledge-based configuration is an Artificial Intelligence technique successfully and frequently applied in different industrial environments (e.g. in the telecommunication industry or in financial services). Informally, configuration can be seen as a special kind of design activity [13], where the final product is built of a predefined set of component types and attributes, which can be composed conforming to a set of corresponding constraints. Configuration systems are of strategic importance for enterprises dealing with highly variant products and services, e.g. response and delivery times to the customer are reduced and invalid orders can be prevented by automatically checking the customer requirements w.r.t. given marketing constraints, technical constraints and constraints related to production processes.

Since the development of the product and the product configurator has to be done concurrently, configurator development time and maintenance time are strictly limited, i.e. the implementation of configuration systems is a critical task and organizations dealing with the provision of highly variant products and services recognize the importance of available measures for analyzing the efforts associated with the development and maintenance of configuration systems.

Effort estimation is a crucial factor when determining the feasibility of a project, creating an offer, or managing resources. As a rule, configuration systems are not standalone systems but have to be integrated into already existing software environments. In this context project managers implementing configuration applications should not be forced to apply additional effort estimation methods but rather be instructed how to effectively apply Software Engineering approaches to knowledge-based systems development. In this paper we show how Function Point Analysis (FPA) can be applied to effort estimation in knowledge-based configuration systems development. FPA is based on a user (requirements) centered view on the software and is platform-independent. The method has first been proposed by [2] with the goal to provide an effort measure for the functional size of software - together with the counting rules it has been adapted several times. Currently it is maintained by the International Function Point Users Group (IFPUG).

In the following we discuss the issue of effort estimation for *developing, maintaining* and *extending* knowledge-based configuration systems. Applying and adapting FPA to configuration software development extends the scope of Software Engineering estimation approaches to knowledge-based systems development. Thus knowledge-based systems development is made transparent within industrial software development processes and effort estimation for traditional software development projects is integrated with effort estimation for knowledge-based software development projects. The remainder of the paper is organized as follows. In Section 2 we discuss and exemplify basic principles of knowledge-based configuration. In Section 3 we show how and under which conditions FPA can be applied to effort estimation in configurator development. In Section 4 we discuss experiences from applying the presented concepts. Section 5 contains related work.

2. Configuration knowledge representation

As pointed out in [16] the modeling of configuration knowledge is a critical task - any framework must address the issues of expressiveness and representational power and

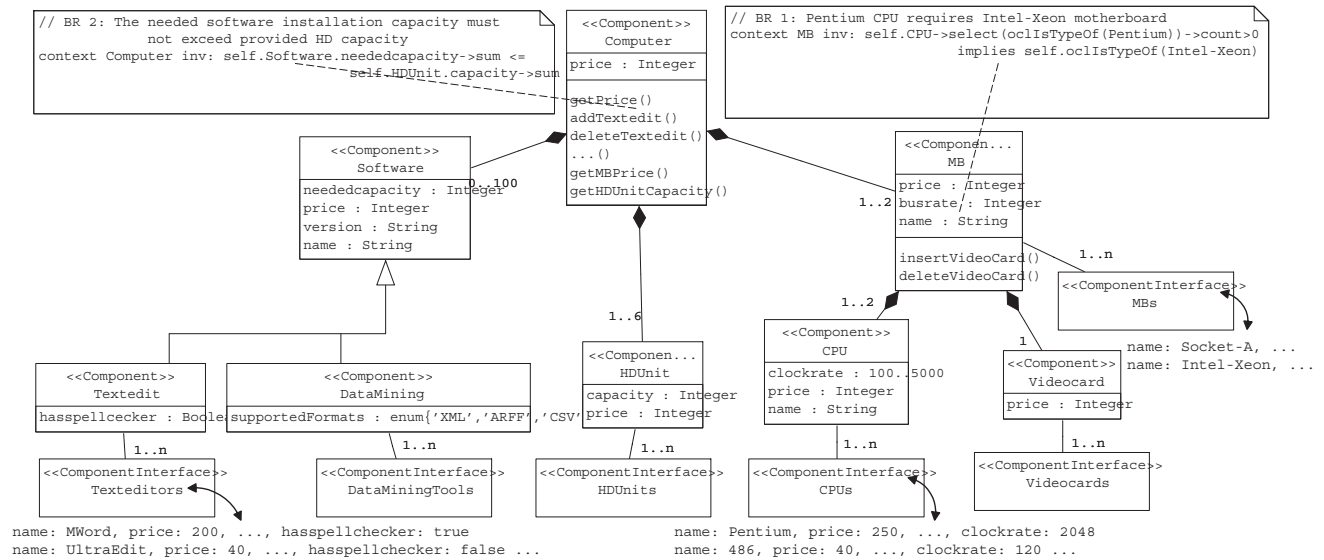


Figure 1. Example configuration model.

provide mechanisms for coping with the high rate at which knowledge changes. In many cases the used description languages for building configuration knowledge bases are not integrated into industrial software development processes. These description languages are difficult to communicate to domain experts which makes it demanding for software development departments to incorporate such technologies into their standard development process. For the realization of configuration systems the Unified Modeling Language (UML, [12]) can be used as notation in order to simplify the construction of a configuration knowledge base [7]. The usage of UML for configuration knowledge representation makes sense for the following reasons:

- UML is widely applied as standard design language in industrial software development.
- UML is extensible for domain-specific purposes, i.e. (using profiles) the semantics of the basic modeling concepts can be further refined in order to be able to provide domain-specific modeling concepts (e.g. modeling concepts for the configuration domain).
- UML has a built-in constraint language (the Object Constraint Language (OCL) [15]). UML and OCL are the perfect combination of representation concepts for designing configuration applications.

In the following the simple UML *configuration model* of Figure 1 will serve as working example. This model represents the generic product structure, i.e. all possible variants of a configurable computer. The basic structure of the product is modeled using component types (basic building

blocks the final product can be built of), generalization hierarchies, aggregations and interfaces to different product catalogs¹. The set of possible products is restricted through a set of business rules (BR1, BR2 in Figure 1) related to technical restrictions, economic factors and restrictions according to the production process. In the literature such a generic description of a product structure is also denoted as *domain description* (DD) [8]. The used modeling concepts are defined in an UML configuration profile [7] and can be interpreted as an ontology in the sense of [5], i.e. ontologies are theories about the sorts of objects, properties of objects, and relations between objects that are possible in a specified domain of knowledge².

Most configuration tasks incorporate additional restrictions (e.g. customer requirements) defining components or attribute settings which must be part of the final configuration. These requirements are called *systems requirements specification* (SRS) [8]. Examples for customer requirements are the following: *set the maximum overall price of the configuration to 1000* or *tell me the price of the actual configuration*. A *configuration result* calculated by a configuration system (configurator) can be interpreted as an instantiation of the configuration model, where all business rules and customer requirements are satisfied. A configuration result can be represented as UML instance diagram [7].

¹Note that not all product catalog instances (related to component interfaces) are shown here completely.

²For a detailed discussion on the modeling elements of a configuration ontology and their translation into an executable representation see [7].

3. Effort Estimation for Implementing Knowledge-based Configurators

There exists a number of approaches investigating the application of Function Point Analysis (FPA) for object-oriented software development (e.g. [9, 14]). However, a direct application to effort estimation in configuration software development results in *significant deviations*. The main reasons for these deviations are the following:

- **Knowledge-based systems development:** existing approaches to FPA (see [1]) do not provide a standard way of accounting for the size of certain types of functional user requirements, notably complex sequences of rules as found in knowledge-based systems.
- **Adjustment factors:** important adjustment factors currently not included in the FPA have to be introduced within the context of knowledge-based configuration. Furthermore, statistical spread resulting from the analysis of empirical data exceeds the standard deviation of FPA adjustment factors, i.e. the calculation of adjusted function points has to be adapted.
- **Counting function points:** based on our experiences in implementing configuration applications in industrial environments, function point values for different complexity classes in configurator projects differ from those in conventional software development. In order to assure appropriate prognoses, function points have to be determined depending on the applied configurator development environment.

Our approach to FPA in knowledge-based configurator development as well assumes a user-centered view on a system. The functionality of the configurator application is defined by the following factors (see Figure 2).

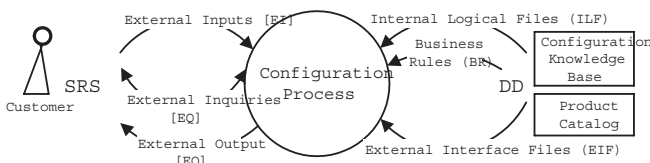


Figure 2. FPA areas

1. **EI - External Input**, i.e. those SRS related to functions which change the actual configuration setting, e.g. inserting a new software component or limiting the maximum price of the overall configuration. Using EI functions a user can add, change and delete basic settings of a configuration.
2. **EQ - External Query**, i.e. those SRS related to functions displaying specific data from the current configuration setting, e.g. the price of a certain CPU part of the actual configuration. External Query (EQ) functions allow users to select and display specific data from configuration settings. For this purpose the user enters selection criteria which are used to match with configuration data, i.e. no data manipulation but a direct retrieval is performed by External Queries.
3. **EO - External Output**, i.e. those SRS related to functions generating output for the user (generation is based on calculations), e.g. the determination of the minimum hard-disk capacity needed for the installation of a certain text editing environment.
4. **ILF - Internal Logical File**. EIs, EQs, and EOs operate on an instance of the domain description (DD). In terms of FPA, the configuration knowledge base is denoted as a set of Internal Logical Files (ILFs), i.e. knowledge elements which are maintained within a configuration application. ILFs allow users to utilize data they are responsible for maintaining.
5. **EIF - External Interface File**. Product catalogs can be seen as an example for External Interface Files (EIF), i.e. knowledge elements which are maintained outside the configuration application. EIFs allow users to utilize data they are not responsible for maintaining (e.g. product data from an external Enterprise Resource Planning system).
6. **BR - Business Rule**. Conventional FPA approaches [1] do not explicitly consider the complexity of business logic - configurators are knowledge-based applications where knowledge complexity has a great influence on development time and costs. In order to consider this important aspect in our estimation approach, we introduce Business Rules as an additional complexity dimension.

In the following EI, EQ, and EO are denoted as *transactional function types*, ILF, EIF and BR are denoted as *data function types*. Figure 3 shows the four steps necessary to determine function points for a configuration application.

1. **Data Functions and Transactional Functions:** ILFs, EIFs, BRs, EIs, EOs and EQs can be directly identified from a given UML configuration model - rules for identifying those units are discussed in Section 3.1.
2. **Complexity of Data Functions:** for each data function, *Record Element Types* (RETs) and *Data Element Types* (DETs) are counted as basic parameters. Based on those parameters the complexity (low, average, high) of each data function can be determined.

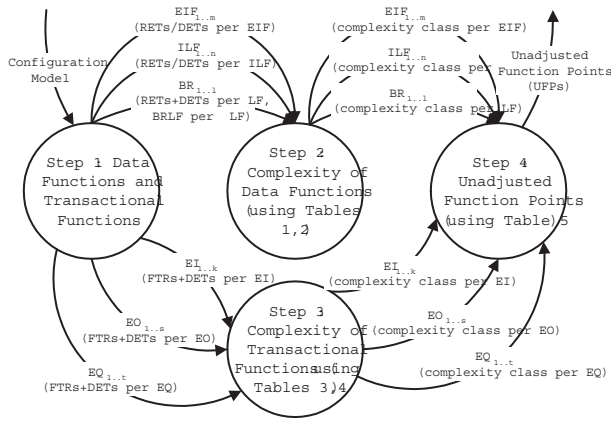


Figure 3. FPA process

3. **Complexity of Transactional Functions:** for each transactional function, *File Types Referenced* (FTRs³) and *Data Element Types* (DETs) are counted as basic parameters for determining the complexity (low, average, high) of the transactional function.
4. **Unadjusted Function Points:** by applying Tables 1-4 the complexity of each data function and each transactional function can be determined. The application of Table 5 results in a value for unadjusted function points (UFPs) for the configuration application, i.e. $UFP = \sum EI_{FPs} + \sum EO_{FPs} + \sum EQ_{FPs} + \sum BR_{FPs} + \sum ILF_{FPs} + \sum EIF_{FPs}$.

3.1. Data Functions

Definition 1: Identification of Logical Files Logical Files (*LFs*) can be identified using the following criteria which are based on a variant of the approach presented in [4]. A logical file (i.e. either an ILF or an EIF) is identified by combining the following two basic rules.

1. Count an entire aggregation structure as a single logical file, recursively joining lower level aggregations.
2. Given an inheritance hierarchy, consider as a different logical file the collection of classes comprised in the entire path from the root superclass to each leaf subclass, i.e. inheritance hierarchies are merged down the leaves of the hierarchy. □

Merging superclasses makes sense since leaf classes with all inherited structures are instantiated during a configuration process. Figure 4 contains an abstract example for the application for the above mentioned rules, i.e. LF_1 represents those classes forming a part of hierarchy, LF_2 and LF_3

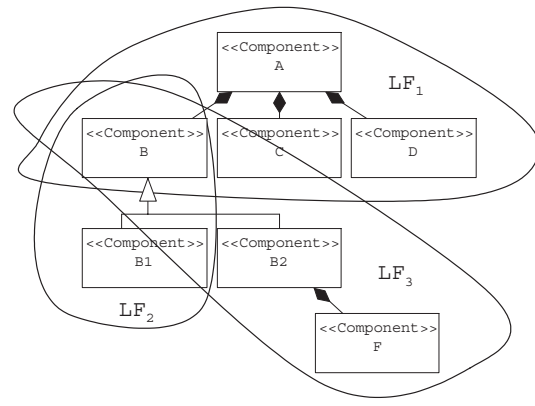


Figure 4. Identification of Logical Files

represent logical files derived from the different paths to leaf subclasses in the generalization hierarchy, where LF_3 also includes the part of relationship between classes $B2$ and F (combination of rule 1 and rule 2)⁴.

Example 1: Identification of LFs In the configuration model of Figure 1 the following LFs can be determined:

- ILFs: {Computer, Software, HDUnit, MB, CPU, Videocard}, {Software, Textedit}, {Software, DataMining}.
- EIFs: {Texteditors}, {DataMiningTools}, {HDUnits}, {CPUs}, {Videocards}, {MBs}. □

The Logical Files identified for the example configuration application are shown in Figure 5.

3.2. Complexity of Data Functions

Definition 2: Complexity of LFs For each LF (ILF and EIF) the number of *Data Element Types* (DETs - unique user-recognizable fields of LFs) and the number of *Record Element Types* (RETs - user-recognizable and logically related data as subgroups of LFs) is computed.

1. Each class within a LF is interpreted as 1 RET.
2. Each attribute within a LF is interpreted as 1 DET.
3. Each involvement of a class in an association with multiplicity > 1 is interpreted as 1 DET within a LF.
4. Each discriminator to a subclass in a generalization hierarchy within a LF is interpreted as 1 DET. □

Depending on the number of RETs and DETs the complexity of ILFs and EIFs can be determined (see Table 1).

³Referenced Logical Files - see Section 3.3.

⁴Note that this is one of several alternatives for the identification of Logical Files (see [4]).

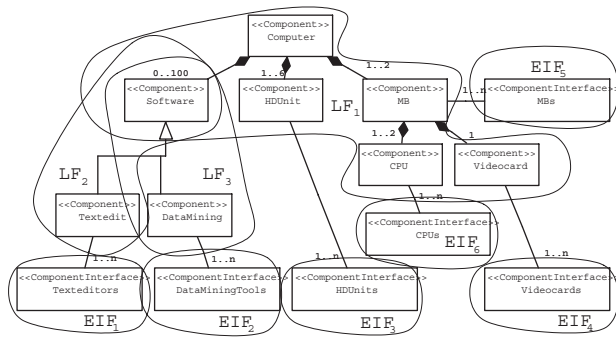


Figure 5. Logical Files of example application

Example 2: Complexity of ILFs and EIFs Based on the entries of Table 1 the data complexity of the computer configuration example can be determined as follows⁵:

- ILFs: $\{\{\text{Computer, Software, HDUnit, MB, CPU, Videocard}\}/[6, 18, \text{average}], \{\text{Software, Textedit}\}/[2, 6, \text{low}], \{\text{Software, DataMining}\}/[2, 6, \text{low}]\}$.
- EIFs: $\{\{\text{Texteditors}\}/[1, 6, \text{low}], \{\text{DataMiningTools}\}/[1, 6, \text{low}], \{\text{HDUnits}\}/[1, 3, \text{low}], \{\text{CPUs}\}/[1, 4, \text{low}], \{\text{Videocards}\}/[1, 2, \text{low}], \{\text{MBs}\}/[1, 4, \text{low}]\}^6. \square$

	1-19 DET	20-50DET	>50DET
1 RET	low	low	average
2-5RET	low	average	high
>5RET	average	high	high

Table 1. Complexity of Data Functions (DF)

Definition 3: Complexity of BRs Depending on the number of RETs and DETs referenced by BRs within a LF and the number of BRs related to a LF (BRLF - BRs per LF), the complexity of BRs is determined (see Table 2). \square

RET+DET	1-4BRLF	5-9BRLF	10-18BRLF
1-16	low	low	average
17-40	low	average	high
> 40	average	high	high

Table 2. Complexity of Business Rules (BR)

This approach provides a measure for the complexity of business rules defined within a LF. In cases, where the limit of 18 BRLF is exceeded, an additional multiplicative factor is added to the determined complexity [6].

⁵We use the notation [#RETs,#DETs,complexity].

⁶We assume that the number of component interface attributes is equal to the number of corresponding component attributes.

	1-4DET	5-15DET	>16DET
0-1FTR	low	low	average
2FTR	low	average	high
>2FTR	average	high	high

Table 3. Complexity of Els

	1-5DET	6-19DET	>19DET
0-1FTR	low	low	average
2-3FTR	low	average	high
>3FTR	average	high	high

Table 4. Complexity of EOs and EQs

Example 3: Complexity of BRs Based on the entries of Table 2 the BR complexity of the computer configuration example can be determined as follows⁷:

$$\{\{\text{Computer, Software, HDUnit, MB, CPU, Videocard}\}_{BR_{1,2}/[10, 2, \text{low}], \{\text{Software, Textedit}\}_{BR_2/[2, 1, \text{low}], \{\text{Software, DataMining}\}_{BR_2/[2, 1, \text{low}]\}. \square$$

Note that $BR_{1,2}/[10, 2, \text{low}]$ in Example 3 is derived by counting RETs+DETs referenced by BR_1 and BR_2 in the corresponding Logical File (LF_1 : {Computer, Software, HDUnit, MB, CPU, Videocard}).

3.3. Complexity of Transactional Functions

The following transactional functions address the user's capability to access configuration knowledge in ILFs and EIFs, i.e. maintaining, putting out and inquiring of configuration process-specific knowledge. In this context LFs are called FTRs (File Types Referenced - see Tables 3 and 4), i.e. a FTR denotes an ILF which is *maintained or referenced* by a transactional function or it denotes an EIF which is *referenced* by a transactional function.

Example 4: Complexity of EI EIs are represented by *addTextedit()*, *deleteTextedit()*, *insertVideocard()*, *deleteVideocard()*. By each of these operations only one FTR exists. For the purposes of our example we assume *low* complexity for each of these operations. \square

Example 5: Complexity of EO EOs are represented by *getPrice()*, *getMBPrice()*. One FTR and one DET is referenced by the method *getPrice()*, i.e. the method has *low* complexity - the same holds for *getMBPrice()*. \square

Example 6: Complexity of EQ EQs are exemplified by *getHdUnitCapacity(HdUnit)*. Two FTRs and one DET are referenced, i.e. the method has *low* complexity. \square

⁷We use the notation [#RETS+#DETS,#BRLF,complexity].

3.4. Unadjusted Function Points

Based on the assignment of function points to different complexity classes (see Table 5) unadjusted function points (UFP) can be determined for the identified data- and transaction functions. These function points represent average values from projects conducted in different application domains using standard configurator development environments. Summing up these function points (see Table 6) re-

Complexity	ILF	EIF	BR	EI	EO/EQ
low	1,5	0,5	1	0,5	0,5
average	2	1	1,5	1	1,5
high	4	2,5	3	2	3

Table 5. Determination of Function Points

sults in 14,5 unadjusted FPs for our example which approximately corresponds to an effort of 1,5 man-months (MMs). This first estimation does not consider special properties of the actual project, i.e. UFPs must be adjusted using a set of adjustment factors (related to general system characteristics - GSC). GSCs are divided into two basic groups⁸.

- *Product characteristics*, i.e. characteristics related to properties of the configuration application (e.g. requirements for distributed configuration support).
- *Project characteristics*, i.e. characteristics related to management strategies and project team (e.g. how well are configuration concepts known by the team?).

	low	average	high	sum
ILF	2	1	0	5
EIF	6	0	0	3
BR	3	0	0	3
EO/EQ	2/1	0	0	1/0,5
EI	4	0	0	2
UPFs				14,5

Table 6. UFPs for example application

Adjusted Function Points (FPs) are determined as follows: $FP = UFP * (0.5 + (TDI * 0.01))$. *TDI* represents the *Total Degree of Influence* calculated from GSCs. Based on this formula efforts related to our example can vary between 0,7 MMs and 4,4 MMs depending on the influence of GSCs.

⁸A detailed overview on different GSCs and their role in knowledge based configuration can be found in [6].

4. Experiences from projects

The development of the presented effort estimation approach has been conducted within the scope of a set of configuration projects in different application domains (e.g. in the banking industry and in the telecommunication industry). We have made excellent experiences in applying the presented estimation concepts - improvements and extensions of the current metrics are still ongoing.

As a rule, early estimations in a project are based on assumptions on the number of classes/attributes/methods, the number of constraints, RETs+DETs accessed by constraints and FTRs+DETs accessed by methods. The full range of presented concepts can be applied when maintaining or extending a configuration application or measuring efforts for an already existing configurator.

Especially within the context of knowledge-based systems development, tool support and experience of the project team play a very critical role. A graphical knowledge acquisition support can significantly reduce the development costs for the configuration system. The same holds for a project team where people already have experiences in developing configuration systems.

The used function point values represent average values for standard configurator development environments. These values are repeatedly improved using data from completed configurator projects. Note that the presented UFPs as well as GSCs degrees of influence are intended as starting point when introducing FPA for estimating configurator application development effort. In order to be more exact and useful, these values have to be adapted for the special purposes of the company, i.e. domain-dependent customizations concerning configurator development environment (UFPs) and product/project characteristics (GSCs) have to be undertaken.

5. Related Work

The identification of sources of variations in effort estimation can significantly contribute to more reliable estimations for software projects [11]. In this paper configurator development is identified as such a source of variation which is tackled by adapting FPA to the special conditions of knowledge-based configurator development. There exists a number of approaches applying FPA to object-oriented software development (e.g. [9, 14]). A direct application of these approaches to configurator development effort estimation results in significant and unacceptable deviations. The COSMIC [1] approach is a ISO standard effort estimation approach within the context of conventional software development projects. Although the method provides an interface for introducing additional measures, COSMIC does not explicitly take into consideration effort estimation

support for knowledge-based systems development. Within the context of our projects we chose to apply conventional FPA, however the integration of our concepts into COSMIC is the subject of future work. The Feature Point approach (see [10]) is an extension to FPA which introduces (beside data functions and transactional functions) the complexity of algorithms as an additional parameter influencing effort estimation. Compared to our approach, Feature Points consider algorithms rather than business rules in knowledge bases - the direct application of this approach as well results in unacceptable deviations. Effort estimation approaches in knowledge-based systems development provide a number of metrics (e.g. size metrics such as rule set density) but do not provide any experimental data to relate metrics to concrete effort sizes. [3] discuss factors influencing development efforts in configurator development - neither relationships to concrete efforts are presented nor explanations are given for the counting approach.

6. Conclusions

We have shown the application and extension of Function Point Analysis (FPA) for effort estimation in the development of knowledge-based configuration systems. A direct application of FPA to configurator development effort estimation results in unacceptable deviations. Consequently, we have adapted the counting of function points to the special conditions of configurator development and extended the FPA approach by special counting rules considering the complexity of business rules.

References

- [1] A. Abran, J-M. Desharnais, S. Oigny, D. St-Pierre, and C. Symons. COSMIC-FFP Measurement Manual. *The COSMIC Implementation Guide for ISO/IEC 19761:2003*, 2003.
- [2] A. Albrecht. Measuring Application Development Productivity. In *IBM Applications Development Symposium*, Monterey, CA, 1979.
- [3] M. Aldanondo and G. Moynard. Deployment of Configurator in Industry: Towards a Load Estimation. In *ECAI 2002 Workshop on Configuration*, pages 125–130, Lyon, France, 2002.
- [4] G. Antoniol, F. Calzolari, L. Cristoforetti, R. Fiutem, and G. Caldiera. Adapting Function Points to Object Oriented Information Systems. In *Advanced Information Systems Engineering, CAISE'98*, pages 59–76, Monterey, CA, 1998.
- [5] B. Chandrasekaran, J. Josephson, and R. Benjamins. What Are Ontologies, and Why do we Need Them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.
- [6] A. Felfernig. Effort Estimation for Knowledge-based Configuration Systems. In University Klagenfurt, editor, *Technical Report KLU-IFI-2003-22*, 2003.
- [7] A. Felfernig, G. Friedrich, and D. Jannach. UML as domain specific language for the construction of knowledge-based configuration systems. *IJSEKE*, 10(4):449–469, 2000.
- [8] A. Felfernig, G. Friedrich, D. Jannach, M. Stumptner, and M. Zanker. Configuration knowledge representations for Semantic Web applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17:31–50, 2003.
- [9] T. Fetcke, A. Abran, and Tho-Hau Nguyen. Mapping the OO-Jacobson Approach into Function Point Analysis. In *Proc. TOOLS '97*, Santa Barbara, CA, 1998.
- [10] T. Hastings. Adapting Function Points to contemporary software systems - A review of proposals. In Australian Software Metrics Association, editor, *Proc. 2nd Australian Conference on Software Metrics*, 1995.
- [11] C.F. Kemerer and B.S. Porter. Improving the Reliability of Function Point Measurement: An Empirical Study. *IEEE Transactions on Software Engineering*, 18(11):1011–1024, 1992.
- [12] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
- [13] D. Sabin and R. Weigel. Product Configuration Frameworks - A Survey. In B. Faltings and E. Freuder, editors, *IEEE Intelligent Systems, Special Issue on Configuration*, volume 13,4, pages 50–58. 1998.
- [14] T. Uemura, S. Kusumoto, and K. Inoue. Function-point analysis using design specifications based on the Unified Modeling Language. *Journal of Software Maintenance and Evolution: Research and Practice*, 13:223–243, 2001.
- [15] J. Warmer and A. Kleppe. *The Object Constraint Language - Precise Modeling with UML*. Addison Wesley Object Technology Series, 1999.
- [16] B. Wielinga and G. Schreiber. Configuration Design Problem Solving. *IEEE Expert/Intelligent Systems and their Applications*, 12,2:49–56, 1997.

Enhancing Mediation Security by Aspect-Oriented Approach *

Li Yang, Raimund K. Ege, Huiqun Yu

School of Computer Science
Florida International University
Miami, FL 33199, USA
{lyang03|ege|yhq}@cs.fiu.edu

Abstract

Research on mediation techniques to integrate data from heterogeneous data sources has made comprehensive progress. However, mediation poses extensive security problems. Protecting proprietary data from unauthorized access is recognized as one of the most significant barriers to the mediation systems. Neither traditional access control methods are adequate to model the flexible access control requirements, nor are they amenable to manage the evolvable security features of mediation systems. This paper addresses to enhancing mediation security by aspect-oriented approach. The basic functionality components and security concerns are separated, independently specified, and then systematically integrated into a unified model. Our approach benefits in both decreasing design complexity of mediation systems and increasing flexibility and dependability of security enforcement.

Keywords: mediation system, security, aspect-oriented, specification

1. Introduction

To gain information from many heterogeneous data sources is the trend for future information system. The mediation [22] task is an extended amalgamation of searching, querying and updating in traditional information systems. Such task can be accomplished via a mediation strategy, i.e. semantic mapping [7, 5] and answering query by source descriptions [21, 11]. In such a mediation strategy, mediators are typically employed to provide an integrated view of information from heterogeneous sources [1, 6]. A mediator provides a mapping of complex models to enable interoper-

ability between clients and sources. One important issue is how to enforce protection for data sources such that every access to a system is controlled, and only those authorized access can take place.

Traditional access control method such as mandatory access control (MAC) and discretionary access control (DAC) [13] are inadequate to reflect the dynamic mediator environment and the flexible access control requirements. Role-based access control (RBAC) [17, 10] models are receiving increasing attention as a generalized approach to access control. Its basic notion is that permissions are associated with roles, and users are assigned to appropriate roles. This greatly simplifies security management. However, security concerns are usually scattered across the entire system, which is difficult to design and manage when the target system is large and complex. A promising new approach to constructing systems with evolvable security features is suggested by the work of Aspect-oriented programming (AOP) [12], which addresses separation of concerns in software development by using specialized mechanisms to encapsulate concerns whose behavior crosscuts essential application functionality.

Inspired by the idea of AOP, we propose a method to enhance mediation security in specification level. Based on our previous work [8, 23], we show how to specify the security concerns and apply them to the mediator modular specification in a uniform way. Datalog is used to specify the mediator functional modules, and first-order predicates are employed to specify the security aspects independently. In logic view, the predicate of security aspects can serve as the condition part of the Datalog specification in mediator, which makes it possible to weave the security aspects into the mediator specification.

The rest of the paper is organized as follows: Section 2 explains our adaptive three-layered mediation framework that features an open adornment-based data model; Section 3 constructs the secure mediation framework via aspect orientation; Section 4 is the conclusion.

*Supported in part by the NSF under grants HRD-0317692 and CCR-0226763, and by NASA under grant NAG 2-1440

2. Our Mediation Framework

2.1. A Three-Layered Mediation Architecture

Our mediator architecture organizes sets of intermediate mediators into layers to handle requests from a user which can be any special device or mobile computing unit (as in Figure 1). The mediator sets will play intermediate roles between users and data sources, to help establish streams to and from the heterogeneous data sources. A user can either query or update heterogeneous data sources.

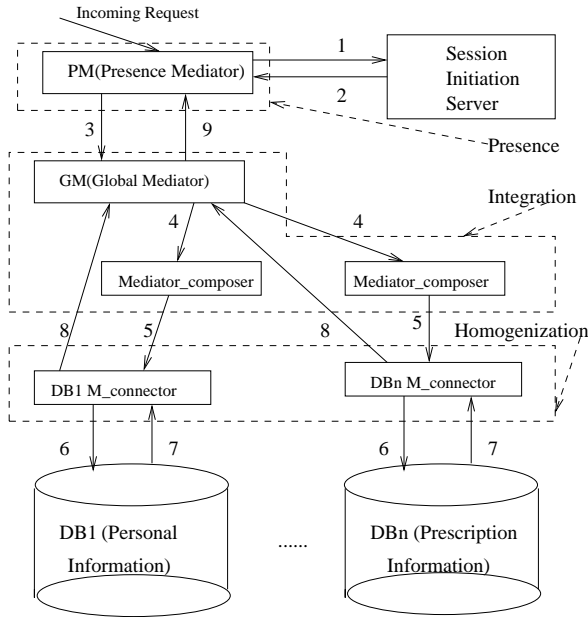


Figure 1. A three-layered mediation architecture

The framework features three layers: presence, integration and homogenization/connector. The upper level is the presence layer that makes the data source seem ever-present to the user and communicates directly with the user. The presence layer is responsible for translating heterogeneous requests from the user into an XML format, extracting the data type of request represented by an XML schema, and translating the response from the XML format into the original user request format. The presence layer makes it feasible that the mediation architecture handles requests from any kind of devices or in any kind of formats via a the two-way format translation. Therefore the work of the underlying layers is encapsulated.

The middle integration layer resolves the schema differences between the user needs and the source availability by schema mapping [3]. The entities in the integration layer are the “Mediator_composers” who are able to decompose the schema if necessary and locate the destination data source for a specific schema via a distributed hash table

algorithm (DHT) [19]. Upon every request from a client, the *session initiation server* coordinates with the “Mediator_composers” group to elect the global mediator. This process of global mediator election dynamically determines the hierarchical structure in the integration layer for each request. This procedure makes the architecture more adaptive to both the network capability and mediator load, and then more efficient for the multimedia data operation, i.e. streaming, than a fixed architecture.

The bottom level homogenization layer contains “Mediator_connectors” that resides on top of actual data sources, and maps the data source schemas to XML schemas. The “Mediator_connectors” stream the actual data or update the underlying data sources upon the request from mediators in the integration layer. They make heterogeneous data sources appear to have a unifying XML schema. As such, we establish an adaptive mediation architecture to handle the two-way data (could be multimedia data) operation (query or update) between the heterogeneous requests and heterogeneous databases.

2.2. Data Model

The primary motivation for mediation technology is to provide support for a broad spectrum of heterogeneous data which are available in different formats. A sound solution to the data integration task requires a clean abstraction of the different formats: any data must be mapped to an *exchange* model from which it is therefore accessible without the use of specific software. Some systems, e.g. SIMS [4] or DISCO [20], or MMM [9], have a fixed application schema like conventional databases. But many systems, e.g. HERMES [2] or Garlic [16], allow for flexible adaptation of the schema as further sources are integrated.

We introduce a *lightweight* exchange model based on XML, enhanced via security (and potentially other) adornments. It is called *the adorned XML model* (AXM). AXM is flexible in data organization, both in the structures that can be described and in the differences in terminology. The security adornment of AXM is essential for the system security. An AXM object has five attributes:

1. *Object ID*. It may be constructed by the mediators to be an expression describing where the object came from. It may also be a pointer to an object in the workspace used to answer the query.
2. *Label* tells what the object represents. Labels are expected to have human-understandable definitions that may be retrieved easily by the user.
3. *Adornment*. Adornment entry identifies the security properties that affect the data processing and system execution. Security adornment indicates a mapping

of principal identities and/or attributes thereof with allowable actions. It is a kind of security policy expression that is often essential in the access control in order to protect resources against unauthorized access. Security adornment plays an important role in the process by which use of resource is regulated according to a security policy and is permitted by only authorized system entities according to that policy.

4. *Type* of its value, either complex type or a simple type like *string*.
5. *Value*, either an atomic value or a set of objects.

With these primitives, it is possible to simulate all the structures that are found in more conventional object-oriented type systems. The adornments can be used not only to define the permissions of the objects in data sources but also to define the roles of the access user.

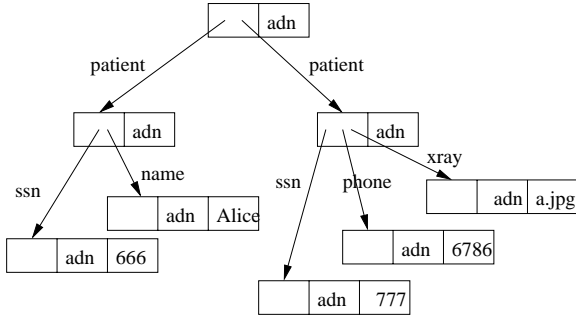


Figure 2. A collection of AXM objects

Figure 2 shows a collection of AXM objects. At the top is a *root* object whose label is *patientGroup*. Its value is a set of *patient* objects, so its type is *complex*. To model semi-structured information sources, we do not insist that data is as strongly structured as in standard database models. For instance, *name* information is sometimes given and sometimes missing.

3. Aspect-Oriented Approach to Enhancing Mediation Security

3.1. The Aspect-Oriented Method

The major issues involved in aspect-orientation for mediation systems are (1) how to separate various concerns, and (2) how to weave aspect models into an integrated system. Issue 1 (*Separation of concerns*) is to separate the functionality modules of mediation systems from the security aspects, which consists of the following steps:

- Identifying basic functionality components in the mediation systems and specifying these components.

- Specifying security requirements.
- Defining the crosscutting section (join points) of the functionality components and security aspects.

Issue 2 (*Aspect weaving*) generates an integrated system by weaving aspect models with the functionality components. The steps include:

- Locating the joinpoints where the functionality components and security aspects interact.
- Defining the behavior of the system in order to enforce security policies on the basic functionality components.
- Integrating aspect models with the functionality components.

3.2. The Component Specification

In our mediation architecture the mediators in the integration layer form the components. Given a set of data sources exported from the homogenization layer, we build mediators to integrate and refine the information. The approach is in the spirit of the declarative specification of mediators in Tsimmis [15]. The query interpretation process is analogous to expanding a query against a conventional relational database view. We will use an example to illustrate the mediator specification and the query interpretation against the mediator specification.

Let us consider two mediators called *med* and *max* that export objects with label *patient*. The *patient* objects fuse information about patients that have the same social security number and are exported by the sources s_1 , s_2 and s_3 . In particular, if source s_1 contains a patient and his name, the exported *patient* object contains the corresponding *name*. If s_2 contains the x-ray examination for the patient and s_3 contains the address information, then the *xray* and *addr* sub-objects are also included in the *patient*. A specification consists of *rules* that define the view exported by the mediator. Each rule consists of a head followed by a : — and a tail. The head describes view objects, whereas the tail describes conditions that must be satisfied by the source objects. In general, the heads and tails are based on patterns of the form $\{<object-id\ adornment\ label\ value>\}$.

The specification of the *patient* object appears in two mediators specification (“MS1” and “MS2”); each rule in the specifications describes the contribution of the sources:

```
(MS1) (R1.1)
<pid(S) (adn R) patient {<(adn R) name N>}>@med :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) name N>}>@s1
  AND role_assign() AND check_permission()
(R1.2)
<pid(S) (adn R) patient {<(adn R) xray X>}>@med :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) xray X>}>@s2
  AND role_assign() AND check_permission()
```

(MS2) (R2.1)

```
<pid(S) (adn R) patient {<(adn R) addr A>}>@max :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) addr A>}>@s3
  AND role_assign() AND check_permission()
(R2.2)
```

```
<pid(S) (adn R) patient {<(adn R) xray X>}>@max :-
  <(adn P) patient {<(adn P) ssn S> <(adn P) xray X>}>@s2
  AND role_assign() AND check_permission()
```

The first rule declares that:

- **if** there is a pair of *binding* s and n for variables S and N (variables are identifiers starting with a capital letter) such that s_1 contains a *patient* top-level object that has a *ssn* sub-object with value s and a *name* subobject with value n , the user was assigned a role after authentication by *role_assign()* predicate, and the data access permission was check against source s_1 by *check_permission()* predicate,
- **then** mediator *med* exports a *patient* object, with object-id *pid(s)*, that has a *name* subobject with value n and a unique system-generated object-id.

The semantics of the second rule in *MS1* (“R1.2”) and in the two rules in *MS2* are defined accordingly. As stated in Section 2.1, each mediator has the potential to be elected as the global mediator and take care of the authentication job.

To illustrate how to interpret the query against the mediator specification, assume that a client wants to retrieve all data of *patient*’s with object-id *pid(‘666’)*. The query can be expressed as:

```
(Q1) <pid(‘666’) (adn R) patient PT> :-
  <pid(‘666’) (adn P) patient PT>
```

The object pattern (or patterns in the general case) that appears in the query tail is evaluated against the object structure of the mediator in exactly the same way that the mediator specification rule tails are evaluated against the object structures of the *mediator_connector*. The object pattern of the query head does not include the usual “@” notation because it is implied that the objects described by the query head refer to the result that will be materialized by the client.

After evaluation the tail of sample query (Q1) against the head of the rules in (*MS1*) and (*MS2*), (Q2), (Q3) and (Q4) are sent to the sources s_1 , s_2 and s_3 respectively.

(Q2)

```
<pid(‘666’) (adn R) patient {<(adn R) name N>}> :-
  <(adn P) patient {<(adn P) ssn ‘666’> <(adn P) name N>}>@s1
```

(Q3)

```
<pid(‘666’) (adn R) patient {<(adn R) xray X>}> :-
  <(adn P) patient {<(adn P) ssn ‘666’> <(adn P) xray X>}>@s2
```

(Q4)

```
<pid(‘666’) (adn R) patient {<(adn R) addr A>}> :-
  <(adn P) patient {<(adn P) ssn ‘666’> <(adn P) addr A>}>@s3
```

The three answer objects received from s_1 , s_2 and s_3 are

then merged into a single *patient* object.

3.3. The Security Aspect Specification

We use RBAC as the underlying security model of mediation systems, and consider two security aspects specification. The RBAC model has the following components:

1. U, R, P and S (users, roles, permissions and sessions respectively), where P is the Cartesian product of operation OP and objects Obj ,
2. $PA \subseteq P \times R$, a many-to-many permission to role assignment relation,
3. $UA \subseteq U \times R$, a many-to-many user to role assignment relation,
4. $user, S \rightarrow U$, a function mapping each session s_i to the single user $user(s_i)$ (constant for the session’s lifetime), and
5. $roles: S \rightarrow 2^R$, a function mapping each session s_i to a set of roles $roles(s_i) \subseteq \{r | (user(s_i), r) \in UA\}$ (which can change with time) and session s_i has the permissions $\cup_{r \in roles(s_i)} \{p | (p, r) \in PA\}$.

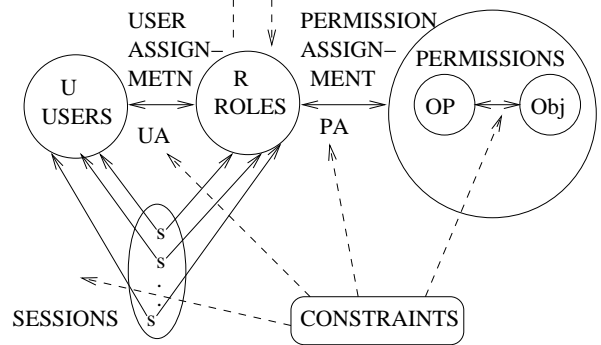


Figure 3. An RBAC model

In RBAC model permissions are associated with roles, and users are assigned roles based on their responsibilities and qualifications. In the mediation system, the security consideration permeates throughout the entire systems. It is possible to express security policies about the system that apply generically to a consideration, and then have the policies be applied through the system automatically. Two security aspects are identified as follows.

Aspect 1. check-role For the separation of duties principle, the same user can not be assigned to any two mutual exclusive roles simultaneously. Mutual exclusion in terms of user assignment specifies that one individual cannot be a member of both roles in exclusive sets. For instance, one

user can not play *patient* and *doctor* role in the same hospital simultaneously. The mediator specification in Section 3.2 only authentication the user and assign role to that user, but the mutual exclusive role checking is void.

Aspect *check_role()* prevents us from assigning the mutual exclusive roles r_1 and r_2 to the same user simultaneously. This aspect can be specified by the following predicate: $\forall u_1, u_2 \in U. (r_1 \in \text{assigned_role}(u_1) \wedge r_2 \in \text{assigned_role}(u_2) \rightarrow (u_1 \neq u_2))$, where $\text{assigned_roles}(u)$ denotes the set of roles assigned to the user u . This aspect is from the constraints on the user assignment.

Aspect 2. check-view In the scenario that the user is allowed to retrieve data *name* and *xray* from source s_1 and s_2 by *ssn* respectively, but the tuple $(\text{name}, \text{xray})$ is sensitive global information. The mediator specifications described in section 3.2 failed to provide the global information filtering mechanism, although they provide the functionality to check the data permission against the data source by the predicate *check_permission()*. When the constraints on the local sources can not protect the information properly by its own, the global level security checking need be enforced into the mediator specifications. It can be used to automatically perform global security checking on sensitive global data retrieval.

Aspect *check_view()* filters the global sensitive data $(\text{name}, \text{xray})$ by checking the mediator views. *Check_view* aspect can be specified by the following predicate: $\forall v, \text{name}_0, \text{xray}_0, \text{ssn}_1, \text{name}_1, \text{ssn}_2, \text{xray}_2. (\neg \text{echo}(v, \text{name}_0, \text{xray}_0) \wedge (\text{echo}(v, \text{ssn}_1, \text{name}_1) \wedge \text{echo}(v, \text{ssn}_2, \text{xray}_2) \rightarrow \text{ssn}_1 \neq \text{ssn}_2))$, where predicate $\text{echo}(v, \text{name}_0, \text{xray}_0)$ denotes that the view v can simultaneously feedbacks the attribute *name*'s value name_0 and *xray*'s value xray_0 , and $\text{echo}(v, \text{ssn}_1, \text{name}_1)$ denotes that the view v can simultaneously feedbacks the attribute *ssn*'s value ssn_1 and *name*'s value name_1 . Similarly does $\text{echo}(v, \text{ssn}_2, \text{xray}_2)$. This aspect is from the constraints on the permission assignment.

3.4. Aspect Weaving

The purpose of aspect weaving is to process the components specification and the aspects specification, and to compose them properly into an integrated specification. Essential to aspect weaving is to specify the join points, where the functionality components and security aspects interact, and to define advices, which encode the appropriate behaviors at the join points.

The component specification language is Datalog and the join points are security-related predicates *role_assign* and

check_permission. The aspect specification language is based on first-order logic. Once a join point is met, there are several types of locations we can operate upon:

- Replacing the join point by well-defined procedure;
- Adding some codes before the join point;
- Adding some codes after the join point.

In our case, we only use the third option, i.e., adding *check_role* after *role_assign*, and adding *check_view* after *check_permission*. In addition to Datalog system, a theorem prover such PVS [14] can be used to automate the query process.

4. Conclusion

This paper proposes an aspect-oriented approach to enhancing security systems. RBAC serves as the security policy model and aspect-oriented approach is applied to isolate, compose and reuse the aspect specification. Two security aspects: *global information leaking* and *mutual exclusive roles* are identified and specified. And the identified weaver can integrates the aspect specification into the mediator specification.

The AOD method provides a rigorous way to identify notions in both functionality and security aspect specifications of mediation systems. The method supports reusable, and reliable design of secure mediator architectures. Security aspects express the essential issues of security requirements and security enforcement mechanisms, and are reusable across different systems. One can express security policies that are intended to be applied across a family of systems as aspects.

Several interesting research topics are: (1) How to properly partition the properties to be modeled by different aspect models, e.g. what information should be included in the base functionality model; (2) What is the dependency between the aspect models; (3) How to coordinate the security enforcement between mediation systems with local DBMSs; (4) How to automate the design process by efficient algorithms and tools.

References

- [1] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Subrahmanian. Query caching and optimization in distributed mediator systems. In *SIGMOD Conference*, pages 137–148, 1996.
- [2] S. Adali and R. Emery. A uniform framework for integrating knowledge in heterogeneous knowledge systems. In *ICDE*, pages 513–520, 1995.

- [3] C. Altenschmidt and J. Biskup. Explicit representation of constrained schema mapping for mediated data integration. In *2nd Workshop on Databases in Networked Information Systems*, number 2544, Aizu, Japan, 2002.
- [4] Y. Arens, C. Y. Chee, C. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1993.
- [5] J. Biskup and D. Embley. Extracting information from heterogeneous information sources using ontologically specified target views. *Source Information Systems archive*, 28(3):169–212, May 2003.
- [6] M. Carey and L. H. et al. Towards heterogeneous multimedia information systems: The Garlic approach. In *International Workshop on Research Issues in Data Engineering: Distributed Object Management*, Taipei, 1995.
- [7] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *The Eleventh International WWW Conference*, Hawaii, US, 2002.
- [8] R. K. Ege, L. Yang, Q. Kharma, and X. Ni. Three-layered mediator architecture based on dht (in press). In *International Symposium on Parallel Architecture, Algorithm and Networks (I-SPAN)*, Hong kong, 2004. IEEE press.
- [9] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y. Ng, D. Quass, and R. D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data Knowledge Engineering*, 31(3):227–251, 1999.
- [10] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.
- [11] A. Y. Halevy. Theory of answering queries using views. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 29(4):40–47, 2000.
- [12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241, pages 220–242. Springer-Verlag, 1997.
- [13] J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. Wiley & Sons, 1994.
- [14] S. Owre, N. Shankar, J. Rushby, and D. Stringer-Calvert. *PVS Prover Guide*. Computer Science Laboratory, SRI International, Menlo Park, CA, September 1998.
- [15] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proc. ICDE Conf.*, pages 251–60, 1995.
- [16] M. T. Roth and P. Schwarz. Don’t scrap it, wrap it! a wrapper architecture for legacy sources. In *23th VLDB Conference*, pages 266–275, Athens, 1997.
- [17] R. Sandhu and E. Coyne. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
- [18] C. Souza dos Santos, S. Abiteboul, and C. Delobel. Virtual schemas and bases. In *Proceedings of the International Conference on Extensive Data Base Technology (EDBT’94)*, Cambridge, UK, pages 81–94. Springer-Verlag, Berlin, 1994.
- [19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service

for internet applications. In *Proceedings of ACM SIGCOMM*, San Diego, August 2001.

- [20] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of disco. In *International Conference on Distributed Computing Systems*, pages 449–457, 1996.
- [21] V. Vassalos and Y. Papakonstantinou. Expressive capabilities description languages and query rewriting algorithms. *Journal of Logic Programming*, 43(1):75–122, 2000.
- [22] G. Wiederhold. Mediators in architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.
- [23] L. Yang and R. K. Ege. Modeling and verification of real-time mediation systems (in press). In *Advanced Simulation Technologies Conference*, Arlington, Virginia, April 2004.

Appendix

A. A Brief Introduction to Datalog

Some standard definitions and terminology on *Datalog* are introduced here.

An *atom* is a formula $p(t_1, \dots, t_n)$, where p is a predicate symbol and each t_i is a term (in the usual first-order logic sense). A term or an atom is said to be *ground* if it is variable free. A *fact* is a ground atom. A *database* is a finite set of facts. A *rule* is a formula written as $A \leftarrow B_1, \dots, B_m$ (in clausal form: $A \vee \neg B_1 \vee \dots \vee \neg B_m$), where A, B_1, \dots, B_m are atoms; A is called the *head* and B_1, \dots, B_m the *body* of the rule. A *logic program* is a finite set of rules together with a database. A *goal* is a formula written as $\leftarrow B_1, \dots, B_m$ (in clausal form: $\neg B_1 \vee \dots \vee \neg B_m$). A *query* is a finite set of rules together with a goal. All variables in rules and goals are implicitly universally quantified.

B. View

The notion of view is particularly important since it is used to consider the same object from various perspectives or with various precisions in its structure (e.g., for the integration of heterogeneous data). We need to specify complex restructuring operations. The view technology developed for object databases can be found in [18].

Declarative specification of a view: Following [18], a view can be defined by specifying the following: (i) how the object population is modified by hiding some objects and creating virtual objects; and how the relationship between objects is modified by hiding and adding edges between objects, or modifying edge labels.

This following example illustrates how to hide the person’s salary by exporting the view (*person*, “high”) from the stored database (*person*, *salary*) by the Datalog. (*person*, “high”) : $\neg(\text{person}, \text{salary}) \wedge \text{salary} > 100k$

Enhancing the Message Concept of the Object Constraint Language

Stephan Flake

C-LAB, Paderborn University, Fuerstenallee 11, 33102 Paderborn, Germany

E-mail: flake@c-lab.de

Abstract

The textual Object Constraint Language (OCL) is an official part of the Unified Modeling Language (UML). A new concept in the recently adopted OCL version 2.0 is the notion of OCL messages that enable modelers to put restrictions on messages sent.

However, this concept shows some shortcomings with respect to the existing OCL language concepts. On the one hand, the proposed syntax does not quite conform to the established notation of OCL. On the other hand, the formal OCL semantics still lacks an integration of OCL messages.

This article reviews the syntax and semantics of OCL messages and presents a new approach to better integrate this concept with the rest of OCL 2.0.

1 Introduction

The Object Constraint Language (OCL) is a declarative expression language that enables modelers to formulate constraints in the context of a given UML user model [12]. Recently, OCL version 2.0 has been adopted by the Object Management Group (OMG) as part of the new UML 2.0 standard [9]. OCL is mainly used to specify invariants attached to classes and pre- and postconditions of operations, but OCL is also applied to formulate well-formedness rules in the metamodel definition of the official UML specification.

As an application example, assume that we have a model with classes `Machine` and `Buffer` and an association between these classes (see Figure 1). The following invariant requires that each instance of class `Machine` has at least one associated buffer:

```
context Machine
inv: self.buffer->notEmpty()
```

The class name that follows the `context` keyword specifies the class for which the following expression should

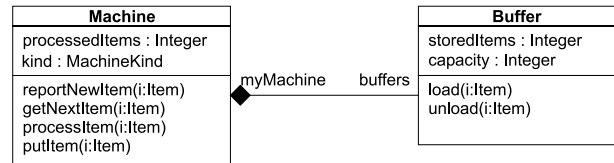


Figure 1. Sample Class Diagram

hold. The keyword `self` refers to each object of the context class. Attributes, operations, and associations can be accessed by dot notation, e.g., `self.buffers` results in a (possibly empty) set of instances of `Buffer`. The arrow notation indicates that a collection of objects is manipulated by one of the predefined OCL collection operations. For example, operation `notEmpty()` returns true when the accessed set is not empty.

In operation postconditions, modelers can put restrictions on messages sent. For example, consider the following requirement for buffer objects.

During execution of operation `load()`, a reporting message has to be sent to the machine to which the buffer belongs.

A corresponding operation specification in OCL may look like this:

```
context Buffer::load(i:Item)
pre: storedItems < capacity
post: myMachine~reportNewItem(i)
```

The expression `myMachine~reportNewItem(i)` is a boolean expression that results in true when at least one message `reportNewItem(i)` has been sent to the associated machine object during operation execution.

However, the syntax and semantics of such message specifications have some significant shortcomings that are discussed in the remainder of this article. In particular, we consider the following issues as being problematic, both in terms of usage by UML modelers as well as w.r.t. the underlying semantics:

```

1: msg.hasReturned() : Boolean
2:   -- Returns true if the message denotes an operation call and if the invoked operation has already returned.
3:
4: msg.result() : OclAny           -- Note: the actual return type is the return type of the invoked operation.
5:   -- Returns the result of the invoked operation if the message denotes an operation call and the invoked
6:   -- operation has already returned. Otherwise the operation returns OclUndefined.
7:
8: msg.isSignalSent() : Boolean
9:   -- Returns true if the message represents a signal.
10:
11: msg.isOperationCall() : Boolean
12:   -- Returns true if the message represents an operation call.

```

Figure 2. Operations on OCL Messages

- Each OCL message expression requires the explicit specification of a destination object.
- The syntax used for message operators, i.e., $\hat{}$ and $\hat{\hat{}}$, is unnecessarily cryptic.
- The expressions for common message specifications are unnecessarily complex.
- Concerning the evaluation of OCL expressions, a message specification can refer to a destination object that might have already been destroyed.
- The OCL semantics description becomes quite complex due to the required data structure that stores the history of messages sent (cf. [4]).

The remainder of this article is structured as follows. In Section 2, we outline the concept of OCL messages as defined in OCL 2.0. Section 3 then discusses the identified shortcomings of the current definition of OCL messages. In Section 4 we then present our redefinition of the OCL message concept. Related work is outlined in Section 5, and Section 6 concludes the article.

2 OCL Messages

Based on the work by Kleppe and Warmer [6, 7], OCL messages have been newly introduced in OCL 2.0 to specify behavioral constraints over messages sent by objects. An OCL message refers to a particular signal sent or a (synchronous or asynchronous) operation called. While signals sent are asynchronous and the calling object simply continues its execution, synchronous operation calls make the invoking operation wait for a return value. An asynchronous operation call is like sending a signal, such that a potential return value is simply discarded. For more details about messaging actions we refer to the action semantics of UML [10, Section 2.24].

2.1 Syntax

The parameterized type `OclMessage(T)` is part of the OCL Standard Library, where the template parameter `T` refers to an operation or signal. A concrete `OclMessage` type is therefore described by (a) the referred operation or signal and (b) all formal parameters of the referred operation or all attributes of the referred signal, respectively. Four operations on OCL messages are predefined (see Figure 2).

OCL messages are obtained by the message operator $\hat{\hat{}}$ that is attached to a *destination object*. For example, the expression `myMachine $\hat{\hat{}}$ reportNewItem(i)` results in the sequence of messages `reportNewItem(i)` that have been sent to the object determined by `myMachine` during execution of the considered operation – recall that the considered expression must have been specified in an operation postcondition. Each element of the resulting sequence is an instance of type `OclMessage(T)`. For example, the exact type of the OCL expression `myMachine $\hat{\hat{}}$ reportNewItem(i)` is `Sequence(OclMessage(reportNewItem(i:Item)))`.

One can make use of so-called *unspecified values* to indicate that an actual parameter does not need to have a specific value. Unspecified values are denoted by question marks, e.g., `myMachine $\hat{\hat{}}$ reportNewItem(? :Item)`.

The *hasSent operator* $\hat{}$ can be used to check whether a message has been sent. This has already been illustrated for the OCL expression `myMachine $\hat{}$ reportNewItem(i)` in Section 1. Note that this operator can easily be derived from the message operator $\hat{\hat{}}$. Each expression of the form `destObj $\hat{}$ msgName(parameters)` can be replaced by `destObj $\hat{\hat{}}$ msgName(parameters)->notEmpty()`.

2.2 Semantics

The OCL 2.0 specification provides two semantic descriptions. The first semantics is a *metamodel-based* approach, i.e., the semantics of an OCL expression is given by associating each value defined in the semantic domain (i.e., the `Values` package) with a type defined in the metamodel (i.e., the `AbstractSyntax` package), and by associating each evaluation with an expression of the abstract

syntax. Given an overall snapshot of the running system, these associations allow to yield a unique value for each OCL expression, which is the result value of OCL expression evaluation. Secondly, a *formal semantics* is defined based on the mathematical notion of an *object model*. This is discussed in more detail in Section 3.

A semantic integration of OCL messages with the rest of OCL is currently only available in the metamodel-based semantics [9, Section 10.2]. In this context, the Values package has a class for *local snapshots*. Local snapshots are kept as an ordered list which allows to access the *history* of the values of an object, e.g., attribute values at the beginning of an operation execution. In particular, local snapshots keep track of the sequence of messages an object has sent. However, there is no dynamic semantics, such that it is undefined which snapshots of a running system are actually stored, i.e., it is not clear *how* local snapshots are created and handled at runtime. Moreover, there is no official formal semantics of OCL messages available, which motivated our previous work [4].

3 Review of the Formal OCL Semantics

The *formal OCL 2.0 semantics* is defined by a set-theoretic approach called *object model* based on work by Richters [11]. The object model of OCL 2.0 is a tuple

$$\mathcal{M} = \langle \text{CLASS}, \text{ATT}, \text{OP}, \text{ASSOC}, \prec, \text{associates}, \text{roles}, \text{multiplicities} \rangle$$

with a set *CLASS* of classes, a set *ATT* of attributes, a set *OP* of operations, a set *ASSOC* of associations, a generalization hierarchy \prec over classes, and functions *associates*, *roles*, and *multiplicities* that give for each $as \in \text{ASSOC}$ its dedicated classes, the classes' role names, and multiplicities, respectively.

In the remainder of this article, we call an instantiation of an object model a *system*. A system changes over time, i.e., the (number of) objects, their attribute values, and other characteristics change during system execution. *System states* keep corresponding information to be able to evaluate OCL expressions. In OCL 2.0, a system state $\sigma(\mathcal{M})$ is formally defined as a triple $\sigma(\mathcal{M}) = \langle \Sigma_{\text{CLASS}}, \Sigma_{\text{ATT}}, \Sigma_{\text{ASSOC}} \rangle$ with the set Σ_{CLASS} of currently existing objects, the set Σ_{ATT} of attribute values of the objects, and the set Σ_{ASSOC} of currently established links that connect the objects.

However, the information stored in this system state triple is not sufficient to evaluate expressions that reason about messages sent; messages are not considered at all in the formal model so far. We therefore added appropriate components to the object model and system states. Thus,

the resulting *extended system state* is a tuple

$$\sigma(\mathcal{M}) = \langle \Sigma_{\text{CLASS}}, \Sigma_{\text{ATT}}, \Sigma_{\text{ASSOC}}, \Sigma_{\text{CONF}}, \Sigma_{\text{currentOp}}, \Sigma_{\text{currentOpParam}}, \Sigma_{\text{sentMsg}}, \Sigma_{\text{sentMsgParam}}, \Sigma_{\text{inputQueue}}, \Sigma_{\text{inputQueueParam}} \rangle$$

that now additionally comprises

- the set Σ_{CONF} of active state configurations over active objects (see [5] for more details about OCL and UML State Diagrams),
- for each currently existing object, the set $\Sigma_{\text{currentOp}}$ of its currently executed operations,
- for each current operation execution, the set Σ_{sentMsg} of sent messages, and
- for each currently existing object, the set $\Sigma_{\text{inputQueue}}$ of received messages that are still waiting to be dispatched.

Parameter values of executed operations and sent/received messages are kept in separate structures for technical reasons. The resulting structure of system states has become comparatively complex, but all listed components are in fact necessary in order to (a) provide a formal semantics for OCL messages and (b) give a dynamic semantics of OCL. We defined a dynamic OCL semantics by means of *traces*, i.e., sequences of system states, based on a set of *noteworthy changes* that identify all changes relevant for the evaluation of OCL constraints [4]. While that work is primarily intended to complete the formal semantics of the OCL 2.0 standard, this article reviews and enhances the concept of OCL messages.

4 Our Approach

To motivate our approach, we first review a message specification found in the OCL 2.0 specification [9, Section 7.7.2]:

```
context Person::giveSalary(amount : Integer)
post: let message : OclMessage = company^getMoney(amount)
      in
        message.hasReturned() -- getMoney was sent and returned
      and
        message.result() = true -- getMoney returned true
```

Unfortunately, this postcondition has a type mismatch; the expression `company^getMoney(amount)` does not return an OCL message, but rather a boolean value, as the `hasSent` operator is applied. We therefore revise the postcondition and use the message operator `^^` to extract the corresponding message(s) sent. Additionally, we adjust the type of variable `messages` to be a sequence of messages:

```

context Person::giveSalary(amount : Integer)
post: let messages : Sequence(OclMessage) =
      company^^getMoney(amount)
in
  messages->forAll(msg:OclMessage | msg.hasReturned())
and
  messages->forAll(msg:OclMessage | msg.result() = true)

```

The postcondition above now requires that *all* messages `getMoney(amount)` sent to object `company` have already returned with result value `true`. But this does not have the originally intended meaning any more. Instead, the actual requirement is that (a) all messages `getMoney(amount)` have already returned and (b) *exactly once* the return result is `true`. Returning `true` stands for getting the money from the company – and the money must not be granted more than once by the company. The correct specification is then as follows.

```

context Person::giveSalary(amount : Integer)
post: let messages : Sequence(OclMessage) =
      company^^getMoney(amount)
in
  messages->forAll(msg:OclMessage | msg.hasReturned())
and
  messages->select(msg:OclMessage | msg.result() = true)
      ->size() = 1

```

The example already exhibits some of the shortcomings of the current approach in OCL 2.0. Firstly, the syntax `^` and `^^` for message specifications easily leads to errors in the specification. The two different operators are very similar in appearance but have totally different results; one denotes a boolean expression, while the other results in a sequence of OCL messages. Secondly, a unique destination object has to be specified together with each referred message. Instead, one might often be interested in a specific message sent to different object (e.g., broadcasts). In such cases a message specification becomes rather complex.

Assume now that a person can have more than one employer, such that `self.companies` refers to the set of `Company` objects that represent the person's employers. In the context of an operation `collectBonus()` that determines the total amount of bonus payments, we require that at least one message `getBonus()` is sent to each employer and that all these messages have returned at the time of post-condition evaluation.

```

context Person::collectBonus()
post: let messages : Sequence(OclMessage) =
      self.companies->collect(c:Company |
        c^^getBonus(self.maritalStatus))
in
  messages->forAll(msg:OclMessage | msg.hasReturned())
and
  self.companies->forAll(c:Company |
    c^^getBonus(self.maritalStatus)->notEmpty())

```

We can directly express the desired, i.e., flattened, sequence of all messages sent to all associated companies with

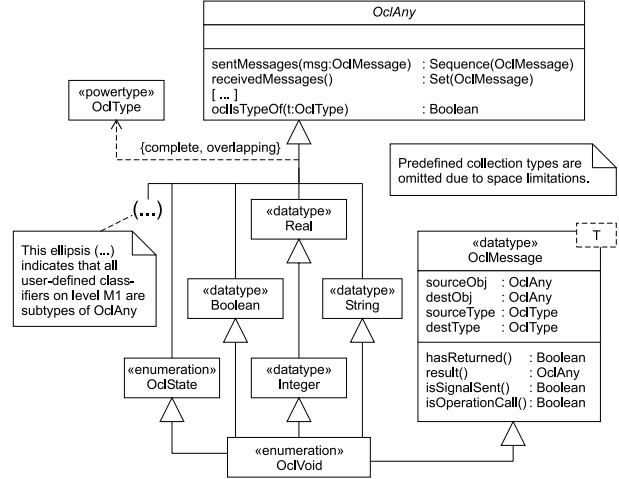


Figure 3. Redefined Type OclMessage

the predefined `collect()` operation.¹ But still, the expression is quite cumbersome to formulate and relatively difficult to understand. For this kind of specification, one would prefer to simply specify the message name without the need to refer to an explicit destination object each time.

4.1 Redefinition of OCL Messages

We suggest a different way to obtain a sequence of messages sent. We define new attributes for type `OclMessage`, i.e., attributes that refer to the source and destination object and to the types of the source and destination object (the latter attributes are for technical purposes as explained in the remainder). The resulting type definition is illustrated in Figure 3. Note that we make use of an enhanced OCL type system that allows to refer to OCL types on the UML user level M1 [3].

With a new operation named `sentMessages()` defined for the general type `OclAny`, which is the supertype of all OCL types, the `collectBonus()` example can then be specified as follows.

```

context Person::collectBonus()
post: let messages : Sequence(OclMessage) =
      self.sentMessages(getBonus(self.maritalStatus))
in
  messages->forAll(msg:OclMessage | msg.hasReturned())
and
  self.companies = messages.destObj->asSet()

```

Firstly, the cryptic and error-prone message operator `^^` can simply be replaced by a new operation on the general supertype `OclAny` as demonstrated above. Secondly, we

¹ Note that one might also assume that the resulting structure is nested, i.e., the result is of type `Set(Sequence(OclMessage))`, but operation `collect` automatically returns the flattened collection. However, as nesting of collections is necessary in many other cases, OCL 2.0 now provides a corresponding operation `collectNested()`.

avoid the explicit specification of a destination object in front of a message declaration. Instead, the new attribute `destObj` for OCL messages leads to a simplified, yet better understandable, formulation of OCL messages, especially in the case of broadcasted and multicasted messages. Moreover, this notation is in line with the established OCL syntax that uses only dot/arrow notation for navigation and applies operation names with arguments. Note here that our formal semantics of OCL messages [4] has only marginally to be adjusted w.r.t. the formal definition of the message tuples.

4.2 Message Destination Objects

A more serious problem arises when a message destination object does not exist anymore at the time of postcondition evaluation. Explicitly referring to such an object in a postcondition does then not make sense. The constraint cannot be evaluated, as the message specification results in an undefined expression. Nevertheless, a message to that object might actually have been sent to that object.

In contrast, our approach captures this situation. We can separately check the value of the attribute `destObj`. If it has the predefined OCL value `OclUndefined` (i.e., the only instance of type `OclVoid`, see Figure 3), the destination object is no longer existing. In fact, this even gives us the chance to explicitly require that certain message destination objects must still exist or must have been destroyed.

Additionally, the attributes referring to the source and destination *types* of messages allow to restrict the kind of participants of message exchanges. For example, we can require that messages `getBonus()` may only be sent to objects of type `Company`:

```
context Person::collectBonus()
post: let messages : Sequence(OclMessage) =
      self.sentMessages(getBonus(?.Status))
      in
      messages->forall(msg:OclMessage |
          msg.destType().oclIsTypeOf(Company))
```

Similarly, we can restrict receptions of messages in preconditions or even invariants. Accessing received messages is discussed in the next section.

4.3 Received Messages

While it is already possible to reason about messages *sent* in OCL 2.0, there is currently no means to access and reason about the messages *received* by an object.

At this point we have to discuss whether it is really necessary to formulate constraints on received messages with OCL. First of all, there are already other UML means to specify behavioral constraints over received messages, e.g., Protocol State Diagrams and Sequence Diagrams. However, it might be necessary to specify *invariants* over received messages that go beyond the specification means of

State Diagrams, e.g., to define a priority scheme after reception of two different signals or to specify a more complex reaction after reception of an external signal. This issue is of particular interest in the domain of embedded real-time systems, where additional real-time properties have to be considered. But as UML and OCL are intended for general purpose modeling, there is no inherent notion of time, such that a dedicated UML profile should be considered in this case.

However, causal relationships concerning requests and acknowledgments might still need to be modeled and are of interest in the context of OCL specifications as well. This soon leads to temporal extensions of OCL that have already been proposed, e.g., in [1]. Unfortunately, such extensions make use of temporal logics to provide a formal semantics, which is definitely out of the scope of the OCL standard in its current state. We therefore stick to our first-order predicate semantics presented in [4]. We make use of the system state component $\Sigma_{inputQueue}$ to provide a semantics for our new operation `receivedMessages()` on type `OclAny` (cf. Figure 3).

Such a semantics is given in the form of a denotational interpretation function $I[[op]](\langle\sigma(\mathcal{M}),\beta\rangle)$ for an operation signature $op = (\omega : t_c \times t_1 \times \dots \times t_n \rightarrow t) \in OP$ over a system state $\sigma(\mathcal{M})$ and an OCL-specific variable assignment β .² In operation signatures, ω is the operation name, c is the class for which the operation ω is defined, and t_c is the respective OCL type. t_1, \dots, t_n are the parameter types, and t is the result type of the operation.

We define the semantics of OCL message operation `receivedMessages()` over a system state $\sigma(\mathcal{M})$ and variable assignment β in the context of a given currently existing object $oid \in \Sigma_{CLASS,c}$. The semantics of operation `receivedMessages()` is formally notated by

$$I[[receivedMessages()]](\langle\sigma(\mathcal{M}),\beta\rangle)(oid).$$

The evaluation result is simply determined by the set $\sigma_{inputQueue,c}(oid)$, where $\sigma_{inputQueue,c}$ is a function over the set $\Sigma_{inputQueue}$ of incoming messages that are waiting to be dispatched. We only have to add the corresponding parameter values stored in $\Sigma_{inputQueueParam,c}$ to each element of $\sigma_{inputQueue,c}(oid)$. However, a detailed formalization is omitted here only for the sake of concision.

We decided that operation `receivedMessages()` returns a *set* of messages rather than a sequence, as the latter would require some kind of ordering predicate on incoming messages. But the order of incoming events is a well-known semantic variation point in UML. One can use the built-in operation `sortedBy()` to induce a sequence of messages if this is desired.

²Variable assignment β determines values for OCL-specific variables, i.e., local variables defined in `let`-expressions and iterator variables used in collection expressions.

5 Related Work

A good overview of approaches that define a semantics for (parts of) different versions of OCL is given in [2]. However, our own recent work [4] so far provides the only formal integration of OCL messages into the rest of OCL.

We know of only one other proposal to enhance the notion of OCL messages, i.e., the work by Kyas and de Boer [8]. They distinguish between local and global specifications for OCL constraints. Additional built-in types such as `OclEvent` with attributes `sender`, `receiver`, and an event kind (`send`, `receive`, `invoke`, `return`) are introduced. Using these types, new predefined attributes `localHistory` and `globalHistory` are presented that allow to access the sequence of sent and received messages. This approach also avoids the rather cryptic message operators $\hat{\cdot}$ and $\hat{\cdot}^{\sim}$. However, an integration into the semantical OCL framework (either the metamodel or the formal semantics) is not described.

In contrast, we can provide a formal definition of our enhancement of the OCL message concept based on the formal notions of our previously proposed *extended object model* and *extended system states*.

6 Conclusion

We identified shortcomings in the syntactical and semantical definition of OCL messages and proposed corresponding enhancements that keep compliant to the established notation and language concepts. Our changes in the definition of OCL messages affect other parts of OCL, e.g., the type system that has to be adjusted to be able to refer to OCL types at the UML user level M1. The OCL community is aware of this problem in the current type system and we expect that this issue will be resolved in the context of the finalization of OCL 2.0.

The formal semantics of OCL 2.0 is relatively complex. However, the underlying logic is still restricted to pure first-order predicate logic, i.e., temporal logic is so far not applied. It should nevertheless be investigated in the future whether temporal logic should be considered both for direct application in user-defined OCL constraints and as an approach to formulate the underlying formal OCL semantics. This could, e.g., avoid the explicit storage of the history of messages sent.

Although there are already some OCL tools available (see <http://www.klasse.nl/ocl> for an overview), there is currently no tool available that supports OCL messages. We hope that our work can influence the development of appropriate tools in the near future.

Acknowledgments. The work described in this article receives funding by the DFG project GRASP within the DFG

Priority Programme 1064 'Integration von Techniken der Softwarespezifikation für ingenieurwissenschaftliche Anwendungen'.

References

- [1] J. Bradfield, J. Küster Filipe, and P. Stevens. Enriching OCL Using Observational Mu-Calculus. In R.-D. Kutsche and H. Weber, editors, *5th International Conference on Fundamental Approaches to Software Engineering (FASE 2002)*, April 2002, Grenoble, France, volume 2306 of *LNCS*, pages 203–217. Springer, 2002.
- [2] M. V. Cengarle and A. Knapp. OCL 1.4/1.5 vs. OCL 2.0 Expressions: Formal Semantics and Expressiveness. *Software and Systems Modeling (SoSyM)*, Springer, 3(1):9–30, March 2004.
- [3] S. Flake. OclType – A Type or Metatype? In *UML 2003 Workshop "OCL 2.0 – Industry Standard or Scientific Playground?"*, October 2003, San Francisco, CA, USA, Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam, The Netherlands, 2003.
- [4] S. Flake. Towards the Completion of the Formal Semantics of OCL 2.0. In V. Estivill-Castro, editor, *27th Australasian Computer Science Conference (ACSC 2004)*, Dunedin, New Zealand, volume 26 of *Australian Computer Science Communications*, pages 73–82, January 2004.
- [5] S. Flake and W. Mueller. Semantics of State-Oriented Expressions in the Object Constraint Language. In *15th International Conference on Software Engineering and Knowledge Engineering (SEKE 2003)*, July 2003, San Francisco Bay, CA, USA, pages 142–149. Knowledge Systems Institute, 2003.
- [6] A. Kleppe and J. Warmer. Extending OCL to Include Actions. In A. Evans, S. Kent, and B. Selic, editors, *UML 2000 - The Unified Modeling Language. Advancing the Standard*. York, UK, volume 1939 of *LNCS*, pages 440–450. Springer, 2000.
- [7] A. Kleppe and J. Warmer. The Semantics of the OCL Action Clause. In T. Clark and J. Warmer, editors, *Object Modeling with the OCL: The Rationale behind the Object Constraint Language*, pages 213–227. Springer, 2002.
- [8] M. Kyas and F. de Boer. On Message Specifications in OCL. In *UML 2003 Workshop on Compositional Verification of UML Models*, October 2003, San Francisco, CA, USA, Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam, The Netherlands, 2003.
- [9] OMG, Object Management Group. UML 2.0 OCL Final Adopted Specification. OMG Document ptc/03-10-14, October 2003. <ftp://ftp.omg.org/pub/docs/ptc/03-10-14.pdf>.
- [10] OMG, Object Management Group. Unified Modeling Language 1.5 Specification. OMG Document formal/03-03-01, March 2003.
- [11] M. Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. PhD thesis, Universität Bremen, Bremen, Germany, 2001.
- [12] J. Warmer and A. Kleppe. *The Object Constraint Language – Getting Your Models Ready for MDA*. Addison-Wesley, 2003.

Entering the Heart of Design: Relationships for Tracing Claim Evolution

Shahtab Wahid, C. F. Allgood, C. M. Chewar, D. Scott McCrickard
Center for Human-Computer Interaction and Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106 USA
{swahid, callgood, cchewar, mccricks}@vt.edu

Abstract

Designers need guidance in tracing knowledge to support the iterative development of interactive software interfaces. Claims show promise in capturing design knowledge with concise descriptions of an artifact's psychological effects on users, but adoptions and modifications made during design processes result in new claims. The manner in which new claims are created based on previous claims establishes unique knowledge relationships not well captured by existing research. This paper proposes six claim relationship types presented with general concepts and examples, allowing a more robust claims analysis process to emerge. The definition of relationships acknowledges claim evolution methods inherent in design, facilitating knowledge reuse and providing structure to advance the science of design.

1. Introduction

The advancement of HCI as a science hinges upon the transfer of knowledge over time within the field. Just as important as the transfer method (design approaches, reuse paradigms, etc.) is the form and structure of the knowledge itself. What better to encapsulate this knowledge than *claims*, design rationale that unapologetically captures both the good and the bad of an artifact? The claim structure proposed by Carroll describes the psychological effects of a designed interface artifact in a usage scenario [1][2][3][10]. Claims address a variety of situational and interface aspects that affect the compatibility of the design and user's models, such as user satisfaction and feeling of reward, color and object layout, and strength of affordances. Inherently objective, claims provide designers with an unadulterated view into what makes an artifact live and breathe, grounded in theories and observations of user experiences.

To illustrate the concept of a claim, we consider a generic timeline artifact that could be used to view all activities and deadlines related to a project. Timelines have been used extensively in information management, resulting in numerous broad statements about their usage

Timelines that dominate the organization, monitoring, and filtering of data . . .

- + add historical context and aid temporal logic by organizing work, correspondence, and transactions in the order that they occur
- + provide a natural guide to experience as a universal skeleton-key
- BUT can subsume metaphors suggested by other interface artifacts and hierarchical categorizations
- BUT may add to confusion by giving an improper timestamp to data with ambiguous temporal characteristics

Figure 1. Claim about a timeline artifact, from [4].

summarized as an example claim in Figure 1. This construct concisely illustrates the tradeoffs of using a timeline with the upsides and downsides of the claim. Through design research and innovation, we try to preserve the upsides and mitigate the downsides.

Therefore, it is important to examine claims as they change and evolve, are created and reused. Just as there are a multitude of human relationships as new generations are born and they themselves reproduce, we propose that there are many claim relationships that exist during the development and evolution of design artifacts. Recording and understanding these relationships provides deeper insight into the overall design process.

Why is this important? Recognizing claim relationship types during the design of an artifact impacts both current and future designs. It enhances the current process by providing a more detailed view of the design history so better decisions can be made during future iterations. Claim relationship types supply valuable knowledge of a claim's origin and development for its potential reuse in another context. In summary, explicit relationships aid in the transfer of claim-embodied knowledge in both the short-term and the long-term.

A need exists for a rich set of classifications for claim relationships. We propose six new claim relationship types in this paper. We also illustrate how our relationship types may come to exist during the development of a system, building from the timeline example above. These

relationship types are by no means exhaustive or all-encompassing. Rather, they are the beginning of a new perspective on the depths of claims and a science of HCI.

2. Related Work

Claims analysis supports the practice of mediated evaluation [9] in human-computer interaction and provides process for evolving a record of design rationale, an argument introduced more than a decade ago [3]. As a form of mediated evaluation, claims analysis blends the benefits of intrinsic evaluation (where a design is described in terms of the performance characteristics it supports) and payoff evaluation (where success in meeting design goals is determined near the end of a project)—it allows explicit and deliberate goal formation, testing, and revision early and often throughout the course of design. As an evolving record of design rationale, the set of claims forming a claims analysis is a series of hypotheses and observations about an artifact in use. While potential benefits have been recognized for making and reusing claims [10], formal and complete guidance for describing relationships among claims is not available.

Claims are one component in Carroll’s task-artifact framework [1] and scenario-based design process [8] that helps designers recognize tradeoffs implicit in the design as users form a goal, act toward its achievement, and evaluate progress. Articulating these tradeoffs as useful generalizations for future design work provides a mechanism for generative problem-solving and design, integrating theory development with design evaluation [3]. Based on the task-artifact framework and the notion of claims reuse, Carroll and Sutcliffe have developed a gradient of progressively powerful object-oriented design analysis techniques whose potential can only be realized with a more clearly defined claim structure [1][10][11].

Certainly, other approaches to design knowledge reuse are prevalent in the software engineering community, especially patterns and object modeling. In our thinking, claims are compatible with both the HCI processes embedded in scenario-based design and patterns records—with *claims as the heart of a pattern (from an HCI perspective) and the focus of usability engineering work*, expressing the key psychological tradeoffs of the reusable artifact modeled by the pattern. Claims-to-pattern relationships are likely to be a many-to-one.

3. Claim Relationship Types

We propose six new claim relationship types that respond to the need for richer descriptions of claim structures and iterative processes within claims analysis. This section defines each relationship type in turn using general concepts, while Section 4 illustrates integrated relationships in a working example.

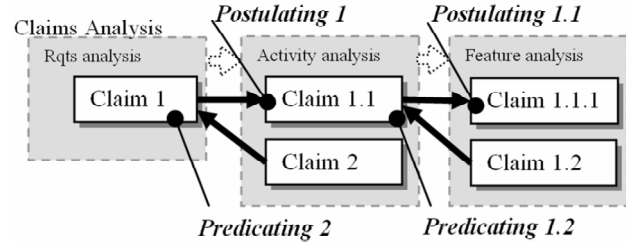


Figure 2. Postulating/Predicating Claims

3.1. Predicating/Postulating Claims

The first key relationship type between claims is the *predication/postulation relationship* apparent in the process of mediated evaluation. In a claims analysis, a designer assigns credit or blame attributions to artifacts, which are continuously refined in subsequent design activities. Design activities typically iterate through three processes, from requirements analysis to general activity design to specific design of features—a pattern paralleled by the themes addressed in each claims analysis. *In each process, a designer collects evidence to assert postulating claims to guide the next process, while alleviating or refuting claims from the previous process with predicating claims based on new ideas or evidence.*

As illustrated in Figure 2, a designer would make Claim 1 to express aspects of the problem domain based on requirements analysis. This leads to the creation of Claim 1.1 as a potentially valuable new user activity through postulation. As specific interface features are conceptualized (Claim 1.1.1) to support the desired user activity (Claim 1.1), Claim 1.1.1 can be referred to as a postulating claim of Claim 1.1. All of the claim upside and downside tradeoffs could be elaborated with scenarios, illustrated with storyboards or other prototypes, and tested with users. Through these design development processes, designers gain inspiration about new ideas—here, an alternate feature (described by Claim 1.2) is found to offer better support for the activity described by the predicating claim, Claim 1.1. Likewise, proposed or validated activity concepts (Claim 2) would be predicated by a claim about the problem domain (Claim 1).

Relating claims in this manner preserves their role within an evolving design rationale context. Recognizing claims in a role as open propositions provides an impetus for continued design development and testing. Alternatively, antecedents or propositions backed by solid evidence suggest a potentially reusable design artifact.

3.2. Executing/Evaluating Claims

Norman presents an argument for interface design as a cognitive engineering discipline, where designers assist the user with progressing through stages of action [7]. He

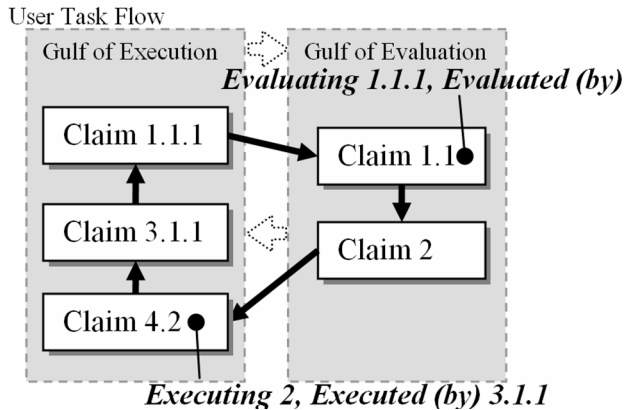


Figure 3. Evaluating/Executing Claims

describes two key hurdles within the stages—crossing the *Gulf of Execution* (after which goals and specific action sequences are decided upon) and the *Gulf of Evaluation* (where the user appraises the current state of a system). Rosson and Carroll’s scenario-based design methodology describes how information design decisions influence the stages of action required for crossing the Gulf of Evaluation, and how interaction design addresses the Gulf of Execution [8]. In information design, interface choices such as use of color, animation, visualization techniques, and layout are made about specific features. Interaction design is more concerned with selection of controls, widgets, affordances, and input techniques.

Certainly, a given artifact may be the subject of both information and interaction claims, and it would be helpful to have a relationship to describe this linkage. Other artifacts may only support the user in one of the Gulfs, but may typically be used with other artifacts that address either the same or opposite Gulf. Therefore, the relationship between two feature claims can be described according to the “destination claim.” *A destination claim in the Gulf of Execution can be the executing claim for claims in either Gulf. Likewise, a claim in the Gulf of Evaluation could be the evaluating claim for other claims in the same or opposite Gulfs.*

The user task flow determines where the execution and evaluation relationships exist between claims. For instance, if a user’s task flow involved Claim 1.1.1, then Claim 1.1, then Claim 2, and so on (see Figure 3), the claim relationships could be described as follows: Claim 4.2 is the executing claim for Claim 2 and is further executed by Claim 3.1.1; Claim 2 extends Claim 1.1 by elaborating evaluation features; and Claim 1.1 is the evaluating claim for Claim 1.1.1. To preserve the context of the task flow, the chain of claims should be related as precisely as possible (for instance, Claim 2 should not be described as an evaluating claim for Claim 3.1.1, without including the intermediate links).

Having a simple vocabulary to describe the relationship of claims across the stages of action and

Gulfs operationalizes Carroll and Kellogg’s notion of “task coverage” [2]. As a heuristic for sufficient detail in a claims analysis, *task coverage* is achieved when at least one claim describes each major artifact state within the task flow across the Gulfs of Execution and Evaluation. In later work, Carroll specifically cautions against replacing a single artifact or claim within a series of task coverage claims, lest the context of task flow be broken [3]. As we move toward developing libraries of claims for reuse, keeping execution and evaluation relationships explicit will preserve task context and assist designers with establishing task coverage in claims analyses.

3.3. Generalizing/Specifying Claims

Claims can have different scopes depending on the granularity of the artifact components which they describe. A general claim might describe psychological effects that result from the holistic design or several distinct portions (combinations of widgets) used in a variety of contexts. General psychological effects can be elaborated by claims that have a narrower scope. These claims apply to very specific parts of an interface (a particular button), usage instances, or user characteristics. They are most useful in guiding component reuse, since they describe an interface at its finest detail and raise in-depth issues related to the interface. However, the “general idea” of a specific claim will often have more frequent applicability to new design problems.

In our framework of claim relationships, the *generalization/specification relationship* is the linkage between two claims with different scopes. *A generalizing claim is the consequence of taking a specific claim and generalizing it to apply to a courser artifact or usage context granularity. A specializing claim is the opposite, in that it is the result of narrowing the scope of a general concept.* The process of generalizing allows one to create claims applicable to many situations (see Claim 2 in Figure 4). This course of action permits one to take ideas from a specific problem and reuse them in a new context to solve design issues—sowing the seeds for innovation and technology transfer. A key concern in generalizing and specifying new claims is with extending or narrowing the scope in an invalid manner, thus, losing the support of empirical or theoretical evidence grounding the original claim. For example, a generalizing claim can only be reliably used in a narrower context, as it inherits upsides and downsides characteristic to specific conditions.

Sutcliffe and Carroll propose a factoring method [10] for evolving between the two types of claims mentioned, although they use the terms “parent claim” and “child claim.” This process involves an analysis of the claim and the situation in which it is used, and allows production of new claims from existing claims. The method is used to examine how a claim’s generalized form spans different

contexts. In the context of this method, since one analyzes a specific claim in order to generate a general claim, the parent is the specific claim and the derived (general) claim is the child claim. Unfortunately, the terms are misleading. With Sutcliffe and Carroll's terminology, a specific claim that leads to the creation of a general claim would be described as "a child spawning a parent." The terms do not distinguish between directions the scope of a claim can change, motivating our argument for the use of generalization and specification relationship types.

3.4. Translating Claims

Existing claims may not be directly applicable to new design problems. *Often though, existing claims provide the basis for the generation of new claims due to recognized similarities between the current problem domain and the one in which the original claim exists. The relationship from the original claim to the new claim is called translation.* Ultimately, claims linked via the translation relationship indicate where cross-domain reuse has occurred in the development of a system (e.g., translation from Claim 1 to Claim 2 in Figure 4).

The crux of translating is the establishment of a correlation between the existing claim and the claim to be created. To accomplish this, the designer is required to consider the existing claim at a deeper level of abstraction, or a generalized version of the claim. While no explicit generalized claim is created, as suggested by Sutcliffe [11], the general form of the original claim exists in the mind of the designer. Then, the specific aspects of the original claim are altered to fit its new context of use, thus creating a new translating claim. Ideally, many of the original tradeoffs will still apply in this new context; however, situating the claim requires re-evaluation of upsides and downsides with respect to this context.

3.5. Fusing/Diffusing Claims

The fusion relationship between claims is the outcome of the combination of two or more claims into a new fusing claim. A developer recognizes that certain aspects of various claims can be applied together in a new and innovative way, such as Claim 3 in Figure 5. The result is a sort of hybrid claim that is pieced together with artifacts and design rationale from each of the supplemental claims. In addition, further design rationale may be required due to novel application of the original artifacts.

Similarly, a designer could break a claim into smaller claims, taking only a fraction of what exists in the original claim to produce a diffusing claim (e.g., Claims 2.1 and 2.2 in Figure 5). This time, the designer focuses on part of a larger claim and elaborates on artifacts and tradeoffs that pertain to the new, smaller claim. This practice may result in the creation of multiple smaller claims,

depending on how the original claim is divided (i.e. there were equal acting parts of the original claim). *This relationship between the original super-claim and the resulting fractional claim is called diffusion.*

Relating claims in this manner can illustrate progress throughout design iterations as well as where claim reuse has occurred. During the design process, testing and evaluation provide the basis for the validation or alleviation of claims. Another result of this process may be the fusion of two claims that seem to demonstrate strong positive results in combination or the diffusion of a claim that exhibits distinctively different results for different aspects of its makeup.

Additionally, two existing claims from completely different problem domains may be fused into a new and innovative claim. This process was noted, but not named by Carroll and Kellogg [2]. An intermediate step, similar to the generalization process described above, requires the designer to consider "what" the claim does, as opposed to "how" this is accomplished. This distinction depends on the level of abstraction at which the claim is considered. In this instance, fusing claims is similar to integration as described by Krueger [6]: the designer "must clearly understand . . . those properties of the artifact that interact with other artifacts." This is accomplished by considering an abstract version of the claim "in which the internal details of the artifact are suppressed."

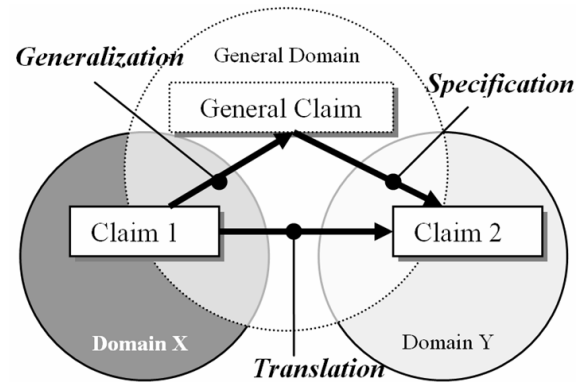


Figure 4. Generalizing/Specifying, Translating Claims

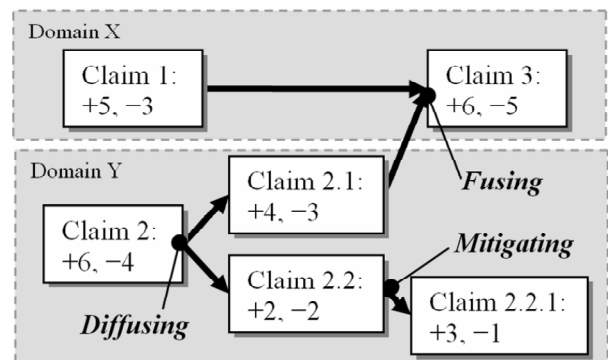


Figure 5. Fusing/Diffusing, Mitigating Claims

3.6. Mitigating Claims

The strength of a claim relies on the explicitness and poignancy of its upsides and downsides. Upsides can represent the potency of an interface, while downsides dictate adverse consequences resulting from the interface design. Explicitly identifying weaknesses of a design often expedites improvement of usability—a process that should be repeated as new flaws are uncovered.

Scenarios are descriptions of a sequence of mental and physical actions a user of an interface may go through. Carroll suggests that one can use scenarios in order to construct new alternative scenarios [1]. The process of analyzing the psychological design rationale within a scenario allows designers to identify alternative scenarios which may be appropriate for other possible usage scenarios. Alternate scenarios are created in a way so that they can handle or correct disadvantages and at the same time maintain or improve strengths of other scenarios.

This same process is valid for claims. *A mitigation relationship is the result of a process in which a new claim is created in order to manage limitations of another claim.* As previously mentioned, claims make their

downsides explicit, clearly identifying areas for which designers must also find solutions. The purpose of a mitigating claim is to resolve the downside in order to improve the overall design (Claim 2.2.1 in Figure 5 removes a downside and gains an upside on Claim 2.2). The method of creating mitigating claims can be repeated as many times as needed until designers are satisfied.

After designers make improvements to an interface in design iteration, usability testing must validate the improvements by testing the performance of the mitigating claims. Thus, mitigating claims become a trace of the design improvements that are made over time.

The repetition of mitigating claim creation and testing for verification produces a chain of mitigating claims. Each claim mitigates a downside in the previous claim. In such a chain, solutions to problems can easily be found, helping general reuse. Typically, the beginning of the chain may contain solutions to slightly more general problems. As more specific problems are identified, mitigating claims find solutions that are more specific. A claim that is further down the chain may turn out to mitigate, not only the claim used to create it, but claims that are even higher up the chain.

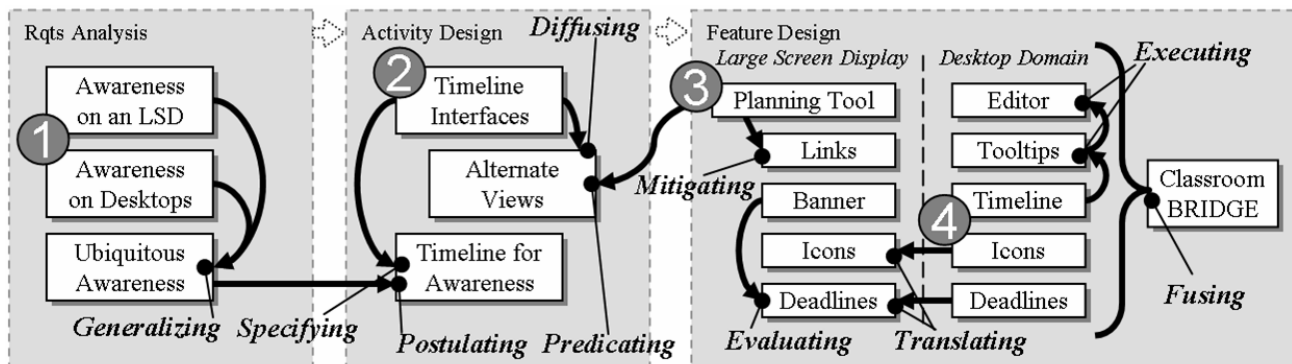


Figure 6. Claim relationships for the ClassroomBRIDGE project; boxes represent claims referenced in Section 4.

4. Example—Claim Relationships in Design

To illustrate how claim evolution takes place in usability engineering efforts, we turn to our development of ClassroomBRIDGE, a collaborative project management tool for middle school science classes [5]. ClassroomBRIDGE built on several previous efforts, both internal to our group and drawn from other researchers, making it rich with examples of claim evolution.

One of the first requirements was developing interfaces that could be used by students at their desk and teachers throughout the classroom. The central technological addition to our suite of classroom tools was a large screen display, positioned at the front of the room. Even though the interfaces were used in different ways—students constantly study the desktop systems, while teachers

quickly get guidance from a large screen—we recognized that both requirements could be expressed as a *generalizing claim* for ubiquitous awareness (see “1” in Figure 6). We *postulated* that using a timeline metaphor for activity awareness supports both user requirements.

In initially brainstorming appropriate activity design approaches, we were intrigued by the Timeline Claim (see Figure 1). We realized that our idea of a timeline metaphor suggested a *specifying claim* regarding the utility of timeline displays expressed in the original Timeline Claim. However, we did not wish to employ the full power of the Timeline Claim as put forth by the authors; instead we created a *diffusing claim* in which we maintain many of the upsides of timelines yet still provide alternate views to the data (see “2” in Figure 6).

Extensive use of a prior, similar system developed by our group revealed limitations in our overall approach—

predicating claim downsides of the alternate view implementation would apply, creating usability concerns. Specifically, the student interface contained a planning tool with the downside: created pages were rarely viewed and never updated after creation. We *mitigated* this downside in our new interface with links in the timeline to the planning tool pages. The timeline links provided a constant reminder of recently added pages, encouraging review and update by the students (see “3” in Figure 6).

In designing specific features (see “4” in Figure 6), we realized that the large screen display would provide teachers with a constant progress view of all student teams on a timeline similar those on student computers. However, our multi-platform system also necessitated that we support many system elements, like the timeline, on both desktop systems and the large screen display. As many elements of the desktop systems were already created and tested, we had to translate much of the information to the large screen display, often reusing elements like the work artifact icons and deadline markers. This was done by making *translating claims* from desktop systems to the large screen for each artifact.

As we discussed previously, the timeline view is not the only view available to students. One challenge in building ClassroomBRIDGE was in connecting the timeline to a concept map, notification banner, chat tool, editor, and other views. Our solutions resulted in numerous *evaluating, executing, and fusing claims*. For example, we color coded related elements in different views to bridge the gap between perception and interpretation in users, two stages in the Gulf of Evaluation and the basis for one of our *evaluating claims*—deadlines were shaded with yellow in both the notification banner and the timeline view. As a second example, to assist users with forming new action plans and initiating execution within the timeline view (Gulf of Execution stages), we implemented tooltips showing authors and dates of work items that would launch appropriate tools when clicked. The tooltip *executing claim* would help a user initiate an update action for a document they recognized to be out of date.

5. Conclusions and Future Work

We have proposed a framework in which claim relations can be named and described as claims evolve over time. A lot of previous work has been done on claims, but little has focused on claim relationships. Our work fills a need for identifying and defining types of claims and links that may exist among this reusable design knowledge.

The primary purpose of such definitions is to make explicit an individual claim’s role within the larger claims analysis and derivation across multiple design studies. Since an interface is the aggregate expression of many claims working together, each claim establishes

relationships with other claims. The six relationships we define allow designers to more richly describe claims in the widest context possible by describing relationships to other claims. By enabling a record of claim evolution, our framework permits one to understand the process used to derive a new claim or reuse a claim in another domain.

Our future work consists of developing a tool based on this framework to organize a claims analysis. This visualization will show all the claims being used in an interface development process along with relationships to claims in a design knowledge repository.

We envision our framework as not only being able to describe *pro forma* design rationale, but to provoke reflection and creative thought processes that would not otherwise be explored by designers. Many of the implicit processes used to generate new claims may be innate for experienced designers, but this formalism will be valuable for design education. With the many benefits of our claim-type definitions, we lay the foundation for a science of design within human-computer interaction.

References

- [1] Carroll, J. M. Making use: a design representation, *Communications of the ACM* 37(12). December 1994.
- [2] Carroll, J. M. and Kellogg, W. A. Artifact as theory-nexus: Hermeneutics meets theory-based design. In *Proceedings of the Conference on Human Factors in Computing Systems* (CHI’89), ACM. New York, 1989. 7-14.
- [3] Carroll, J. M., Singley, M. K., and Rosson, M. B. Integrating Theory Development with Design Evaluation, *Behavior and Information Technology* 11, 1992.
- [4] Freeman, Eric and Gelernter, David. Lifestreams: A Storage Model for Personal Data. *SIGMOD Record* (ACM SIG on Management of Data 25(1), 1996.
- [5] Ganoe, C., Somervell, J., Neale, D., Isenhour, P., Carroll, J. M., Rosson, M. B., and McCrickard, D. S. Classroom BRIDGE: Using Collaborative Public and Desktop Timelines to Support Activity Awareness. In *Proceedings of the ACM Conference on User Interface Software and Technology* (UIST ’03), Vancouver BC Canada, Nov 2003.
- [6] Krueger, Charles W. Software Reuse, *ACM Computing Surveys (CSUR)* 24(2), June 1992.
- [7] Norman, D. A. (1986). Cognitive engineering. In D. A. Norman & S. W. Draper, Eds. *User Centered System Design*, 31-62, Hillsdale, NJ: Erlbaum.
- [8] Rosson, M. B. and Carroll, J. M. (2002). *Usability Engineering: Scenario-Based Development of Human Computer Interaction*. Morgan Kaufmann Publishers.
- [9] Scriven, M. (1967). The methodology of evaluation. In R. Tyler, R. Gagne, & M. Scriven (Eds.), *Perspectives of curriculum evaluation*. Rand McNally, 39-83.
- [10] Sutcliffe, Alistair G. and Carroll, John M. Designing claims for reuse in interactive systems design, *International Journal of Human-Computer Studies* 50. 213-231, 1999.
- [11] Sutcliffe, Alistair. On the effective use and reuse of HCI knowledge, In *ACM Transactions on Computer-Human Interaction*, 7(2), 197-221, June 2000.

Future Proofing and Retargeting Application Logic Using O2XML

Marselina Wiharto and Peter Stanski

School of Computer Science and Software Engineering, Monash University, Australia

Marselina.Wiharto@csse.monash.edu.au, Peter.Stanski@stanski.com

Abstract. The software engineering tasks associated with software retargeting to new languages and future platforms are formidable when performed by hand. In this paper, we present one possible way that may be used to optimise the porting process. Through the usage of eXtensible Markup Language (XML) technologies and compiler code generator backend, we present a framework for rapid application logic transformations and its abstraction from programming languages.

1. Introduction

In the recent years, we have seen a greater convergence on core software industry platforms into J2EE and .NET technologies. Whilst these technologies are relatively young, they often require the interoperability with more mature software investments as parts of a complete solution.

Some of the older technologies now need to be either migrated or made to interoperate with more recent technologies. The interoperability (interop) path is more appealing from a project delivery timeframe perspective. However, some older technologies are unable to interoperate without significant reworking. Thus, this is one of the main appeals for XML Web Services, which can abstract and hide legacy components. However, once these systems are made to interoperate, they frequently suffer from scalability and reliability issues. This is usually attributed to those systems never being designed for the higher levels of demand.

The second more expensive option is that of application porting, or more precisely the extraction of business rules and functions, and their rewriting in a more recent programming language. Some software automation conversion tools are available to help, however these struggle with semantic mappings from one language to another. Thus, automation can only port parts of the application logic and may itself introduce hard to find bugs during the mapping operations.

In an ideal world, if all application logic was stored in a semantically rich source code format, this would simplify the migration and retargeting automation

process. New versions of the same business rules could be generated (stamped out) in other programming languages by migration tools without any manual porting activities, which are time consuming.

With this in our minds, we set out to investigate this proposition. Thus, the above model was used as the basis for our prototype system described herein.

2. Introducing O2XML

Object Oriented XML (OO XML), or O2XML in brief, is a language neutral XML based markup language. O2XML is aimed at representing OO languages in general. However, our current version of O2XML focuses on representing C#, J#, Java, and VB .NET source codes.

Source codes, which are used to express application logic, can be broken down into smaller components. For example, a class definition can be made up of variable declarations and method declarations and definitions. A method definition can be decomposed further into even smaller constructs, such as assignments, If statements, For loops, and method invocations.

In O2XML, these constructs are represented using elements with or without attributes. The current O2XML schema supports 27 fundamental constructs ranging from class declarations and definitions down to If statements and binary expressions.

The following Code Listing 1 presents an example O2XML document: SimpleMathematics.xml, which represents the SimpleMathematics class. This class has been simplified to include only one method, which is the Square function. This function takes an integer as an argument and returns the result of multiplying this integer with itself.

As can be seen from this O2XML document, a class is represented by the *class* element with the class name stored in the *name* attribute of the *class* element. The *class_definition* element houses the *methods* element, which in turn consists of a *method* element. This *method* element represents the Square function. Embedded within the *implementation* element of the *method* element is a *return_statement* element, which returns the result of the self-multiplication of the method's integer parameter.

```

<?xml version="1.0" encoding="UTF-8"?>
<o2xml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="o2xml.xsd">
<namespace>Demo</namespace>
<class name="SimpleMathematics">
<class_definition>
<methods>
<method name="Square" return_type="int" isPublic="true">
<parameters>
<parameter name="a" type="int" arrayDimension="0"/>
</parameters>
<implementation>
<return_statement isString="false">
<calculation arithmetic_operator="multiply">
<value>a</value>
<value>a</value>
</calculation>
</return_statement>
</implementation>
</method>
</methods>
</class_definition>
</class>
</o2xml>

```

**Code Listing 1: Sample O2XML Document:
SimpleMathematics.xml**

3. Prototype System

The main functionality of the prototype system is to produce source codes and executables in multiple languages from any given O2XML document. The current prototype system includes the O2XML and two components that produce C#, J#, Java, and VB .NET source codes and executables. The functionality of these two components overlaps but they are based on two different technologies. The first component uses the eXtensible Stylesheet Language Transformation (XSLT) [6] approach and the second component uses the CodeDOM technology [5].

XSLT. XSLT is an XML based technology. It is an XML based scripting language that can be used to transform XML documents to other document formats, such as text and HTML [6].

CodeDOM Technology. The CodeDOM technology is a subset of the Microsoft .NET Framework. The core of this technology is the System.CodeDom [9] and the System.CodeDom.Compiler [10] namespaces. These namespaces work together to create a language independent in-memory representation of source codes and produce source codes as well as executables (DLLs and EXEs) in different supported languages from this representation [5].

In the CodeDOM world, each source code component is represented by an object. For example, a class is represented by a CodeTypeDeclaration object, while an If statement is substituted by a CodeConditionStatement object. Similar to source code structure, these objects can also be nested, such that a CodeConditionStatement

object will be indirectly housed in a CodeType Declaration object.

3.1. XSLT Transformer

The XSLT transformer component relies heavily on the XSLT scripts to transform O2XML documents into language specific source codes. The number of XSLT scripts required corresponds directly to the number of languages, in which source codes can be produced. The current XSLT transformer is capable of outputting C#, Java, and VB .NET source codes. The core of the XSLT transformer component includes three XSLT scripts, i.e. C#, Java, and VB .NET XSLT scripts.

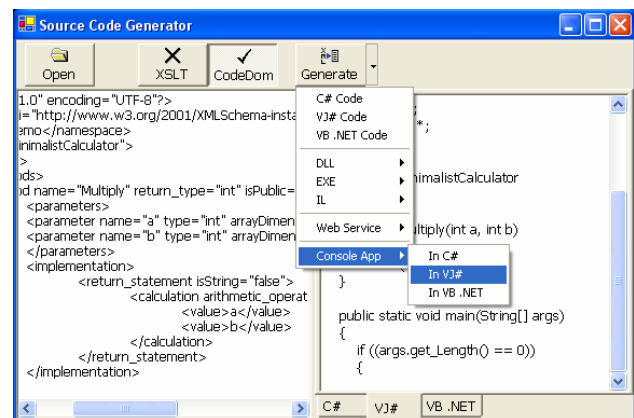
The XSLT transformer does not directly output executables. However, a simple tool that compiles the outputted source codes can be written to allow the XSLT transformer to also output executables.

3.2. CodeDOM Generator

As reflected in its name, the CodeDOM generator is based on the CodeDOM technology. This generator is able to produce more outputs in comparison to the XSLT transformer. It is capable of outputting C#, J#, and VB .NET source codes, DLLs, EXEs, ILs, .NET XML Web Services, and console applications from any given O2XML document.

The last two output types (i.e. .NET XML Web Services and console applications) are essentially variations of the basic DLL and EXE outputs. At the core of a .NET XML Web Service is a DLL and a console application takes the form of an EXE.

These various prototype components are put together to constitute the prototype system with a graphical user interface as shown in Figure 1 below.



**Figure 1: The Graphical User Interface of the
Prototype System**

Examples of a .NET XML Web Service and a console application generated (using the prototype system) based on the sample O2XML document in Code Listing 1 are shown respectively in Figure 2 and Figure 3. The .NET XML Web Service has been generated in C#, while the console application is a VB .NET executable (EXE).

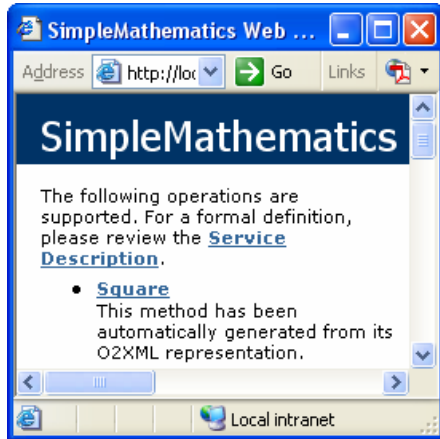


Figure 2: SimpleMathematics .NET XML Web Service in C#

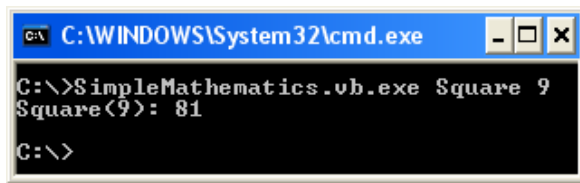


Figure 3: Command Line Execution of the Square Function of the VB .NET SimpleMathematics Console Application

4. Related Works

The idea of representing source codes using an XML based markup language is not new. A number of research projects has introduced several XML based markup languages to represent source codes of languages, such as C++ and Java. Despite their various motivations, the following research projects are presented to illustrate the different XML based markup languages and how they have been exercised.

4.1. JavaML

Badros [4] from University of Washington proposes the use of Java Markup Language (JavaML) to represent Java source codes. He claims that by having such a representation, programs can work with source code without requiring a parser. Instead, XML tools can be used to manipulate the corresponding XML based

representation of the source codes, i.e. JavaML. In his paper [4], he demonstrates the parsing of Java source codes to produce JavaML and the transformation of JavaML to obtain Java source codes back. He also shows that source code statistics and HTML documentation can be produced from JavaML.

4.2. srcML

Maletic and colleagues [7] propose a similar XML based markup language, which they named SouRce Code Markup Language (srcML). Despite its generic name, the current implementation of srcML focuses on giving structures to C++ source codes. In another paper [8], they outline a lightweight C++ fact extractor, which is of a parallel concept with the source code statistics produced by Badros. This extractor obtains information such as the number of occurrences of certain constructs from the srcML representation of the C++ source codes.

4.3. JavaML, PascalML, and PLIXML

Another similar research project to ours is by McArthur and colleagues [3] who present three XML based markup languages: JavaML, Pascal Markup Language (PascalML), and PL/IX Markup Language (PLIXML). These markup languages are used to represent Java, Pascal, and PL/IX source codes respectively. In their paper, they focus on PLIXML and the possibility to use PLIXML to help migrate legacy applications written in PL/IX to a more recent programming language.

4.4. CppML, JavaML, and OOML

In University of Waterloo, Mamas and Kontogiannis [2] put forth three XML based representation as part of their software engineering environment. They use C++ Markup Language (CppML) to represent C++ source codes and JavaML to represent Java source codes. Subsequently, they represent CppML and JavaML using OO Markup Language (OOML) by transforming them to OOML using XSLT. By doing this, they only need to develop a single component to analyse both C++ and Java source codes as they are uniformly represented by OOML.

5. Evaluation

In comparison with the related research above, O2XML is different as it attempts to represent more than one programming language. Moreover, our prototype system has been distinctly developed to generate source codes as well as executables in multiple OO languages, which is not demonstrated by any of the research project previously presented. Additionally, the prototype system

also generates .NET XML Web Services and console applications as an example of retargeting application logic to different types of components in different languages.

In this section, we present the evaluation of the prototype system in two main aspects, which are the size of the O2XML documents compared to the size of the corresponding source code files generated and the performance of the XSLT transformer and the CodeDOM generator in producing source codes and executables (DLLs and EXEs).

This evaluation was performed on 15 sample O2XML documents that represent calculation as well as sort and search algorithms. The size of these files ranges from 1,246 to 39,729 bytes.

A total of 10 test runs were carried out. Using the 15 sample O2XML documents, in each run, C#, Java, and VB .NET source codes (15 files each) were generated using the XSLT transformer and C#, J#, and VB .NET source codes, DLLs, and EXEs (15 files each) were generated using the CodeDOM generator.

5.1. Size Comparison

The size comparison between O2XML documents and their corresponding source code files can be grouped into two sets of graphs. The first set of graphs contrasts the size of the O2XML documents against the size of the XSLT generated C#, Java, and VB .NET source code files. The second set of graphs evaluates the size of the O2XML documents against the size of the CodeDOM generated C#, J#, and VB .NET source code files.

These graphs are depicted respectively in the following Figure 4 and Figure 5. The comparison is measured in byte figures, which are obtained from averaging the byte sizes of the relevant files.

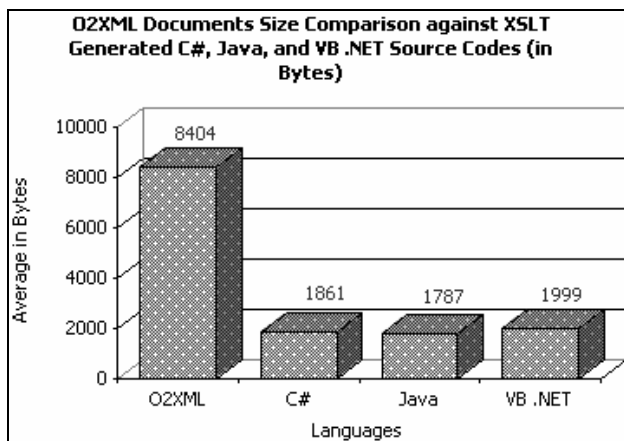


Figure 4: Size Comparison between O2XML Documents and XSLT Generated C#, Java, and VB .NET Source Code Files

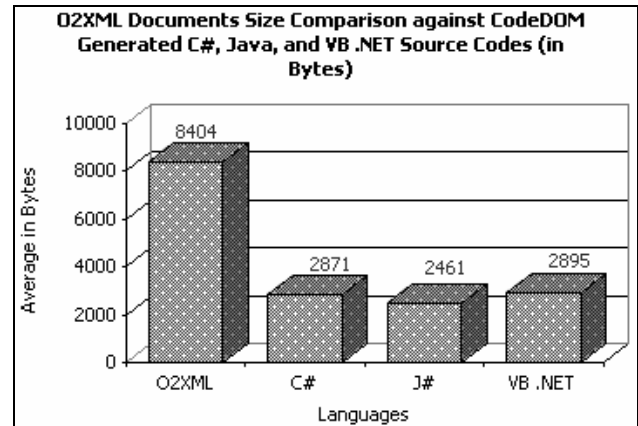


Figure 5: Size Comparison between O2XML Document and CodeDOM Generated C#, J#, and VB .NET Source Code Files

These two graphs show that the size of the O2XML documents is on average 2.9 to 4.7 times larger than the size of the corresponding source code files. Meanwhile, the trend across the languages is the same within the two graphs. Java/J# source codes have the lowest average byte size, while VB .NET source codes have the highest average byte size.

Contrasting the two sets of graphs, the CodeDOM generated source codes have higher average byte sizes compared to the XSLT generated source codes. This is because the CodeDOM generator uses white spaces to align the source codes, while the XSLT scripts uses tab characters to help format the source code alignment. Another reason is that the CodeDOM generator also outputs additional new line characters when it generates constructs such as nested binary expressions.

5.2. Performance Evaluation

Two performance evaluations were carried out. The first evaluation compares the performance of the XSLT transformer with the performance of the CodeDOM generator in producing C#, Java/J#, and VB .NET source codes. The second evaluation focuses on the performance of the CodeDOM generator in outputting C#, J#, and VB .NET DLLs and EXEs.

The test runs for the performance evaluation were executed on a machine with the following specifications:

- CPU Speed: Pentium IV CPU 2.40 GHz
- RAM Size: 480MB RAM
- Operating System: Microsoft Windows XP Professional, Version 2002 Service Pack 1

The results that compare the performance of the XSLT transformer and the CodeDOM generator in producing C#, Java/J#, and VB .NET source codes are given in Figure 6 in millisecond measurement.

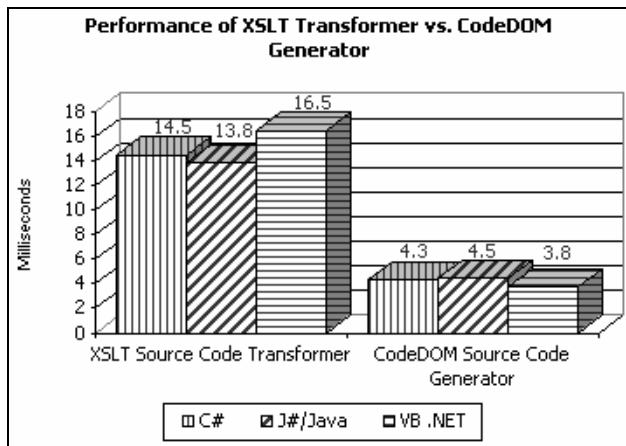


Figure 6: Performance of XSLT Transformer vs. CodeDOM Generator in Outputting C#, Java/J#, and VB .NET Source Codes

The graph above shows that the average time taken to output C#, Java, and VB .NET source codes using the XSLT transformer only varies around 2 to 2.7 milliseconds. Using the CodeDOM generator, the variation is down to less than one millisecond (0.2 to 0.7 millisecond).

Comparing the performance of the XSLT transformer against the CodeDOM generator, however, indicates a more significant difference. The XSLT transformer takes on average 3.6 times longer to generate the source code files.

It needs to be noted that the performance figure of the CodeDOM generator in the above graph does not include the warm up overhead. A warm up overhead is the extra time required to generate the first source code file using the CodeDOM technology. This extra time is mainly dedicated to initialise the various CodeDOM objects that are required to represent the source code components, and for the .NET runtime environment to JIT compile them.

If the warm up overhead is taken into account, the average time required to generate the source codes can go up to 31 milliseconds. This, however, only alters the graph of the language, in which the first source code file was generated. In our test runs, the first source code output was in C#. With the C# average being 31 milliseconds, the overall averages of time taken to generate source codes using the XSLT transformer and the CodeDOM generator are fairly close (XSLT transformer: 14.9 milliseconds to CodeDOM generator: 13.1 milliseconds on average).

In general, both generator components require a reasonably short amount of time to generate the source codes, i.e. a maximum average of 31 milliseconds, which is less than a twentieth of a second.

The next graph in Figure 7 presents the performance of the CodeDOM generator in outputting DLLs and EXEs measured also in milliseconds.

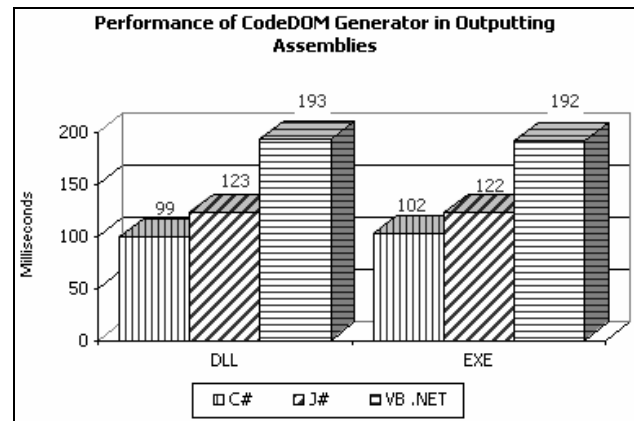


Figure 7: Performance of CodeDOM Generator in Outputting C#, J#, and VB .NET DLLs and EXEs

From the graph, it can be noted that across the three different languages, there is a similar trend with C# DLL/EXE generation taking the least amount of time and VB .NET DLL/EXE generation taking the most amount of time. In general, using the CodeDOM generator, it takes less than one fifth of a second to generate a DLL or EXE from an O2XML document.

In summary, although the O2XML documents are generally larger in size compared to their corresponding source code files, the prototype system (i.e. the XSLT transformer and the CodeDOM generator) is still capable of outputting source codes, DLLs, and EXEs from these documents in a considerably short amount of time.

6. Future Works

The current prototype system still has plenty of opportunities for future improvements. To improve the current prototype system, the following extensions can be introduced.

6.1. Source Code to O2XML Parser

Having a parser to obtain O2XML documents from their originating source codes would make our prototype system more efficient. This extension would complete the prototype as an end-to-end generator that is capable of translating source codes from one language to many other languages.

6.2. Thorough Constructs Support

The current O2XML schema does not provide support for every possible OO language construct. The schema can be extended further to support all OO language constructs, including jagged arrays, unlimited multidimensional arrays, event handlers, delegates, and inner classes.

6.3. Further Language Support

The above extension can be complemented by adding more language support. For example, constructs of OO languages such as Eiffel and C++ could be supported by the O2XML schema, while the prototype system could be updated to produce Eiffel and C++ source codes and executables.

Currently, support to output COBOL .NET source codes are being added to the prototype. However, this still remains unstable for the generation of any substantial application logic.

6.4. Cross Language Library Mappings

Cross language library mappings can be done by mapping the APIs of one language to the other. To add this feature to the current prototype, the Java APIs need to be mapped to the .NET APIs. In having this feature, it would allow for a seamless source code generation across all the currently supported languages. However, we would not expect perfect mappings to take place. Hence, occasionally, some manual mapping might be required.

6.5. Longhorn Integration

The key component of the Microsoft Longhorn Operating System technology is its new eXtensible Application Markup Language (XAML). XAML is an XML based markup language that can be used to define application user interfaces independent of the programming language used [1]. Thus, a single XAML user interface definition can be reused by backend processors (i.e. which implement event handling logic) written in different .NET supported languages.

The current prototype system can be improved by integrating XAML with O2XML. Such integration would mean that all application development could now be done in pure XML.

7. Conclusion

Having presented the related works and outlined the significance of XML, we have found that the other projects have hinted at numerous source code processing possibilities.

In our work, we have successfully transformed O2XML representations to other languages and managed to construct functional applications and services.

We have found that both XSLT and CodeDOM are suitable candidates as transformers. Since the CodeDOM also functions as a backend code generator, our system

has essentially created an XML input API for the generation of source codes and binaries. Thus, our project theoretically has the ability to cookie stamp application logic, provided the application source code is represented using the semantically rich O2XML format.

References

- [1] B. Rector (2003), "Chapter 1: The "Longhorn" Application Model", MSDN Library, October 2003 [Online], Available: <http://msdn.microsoft.com/longhorn/default.aspx?pull=/library/en-us/dnintl/long/html/longhornch01.asp>
- [2] E. Mamas and K. Kontogiannis (2000), "Towards Portable Source Code Representation Using XML", *Proceedings of the Seventh Working Conference on Reverse Engineering*, IEEE Computer Society Press, Brisbane Australia, 23 – 25 November, pp. 172 – 182.
- [3] G. McArthur, J. Mylopoulos, S.K.K. Ng (2002), "An Extensible Tool for Source Code Representation Using XML", *Proceedings of the Ninth Working Conference on Reverse Engineering*, IEEE Computer Society, Richmond, Virginia, USA, 29 October – 1 November, pp. 199 – 208.
- [4] G.J. Badros (2000), "A Markup Language for Java Source Code", *Proceedings of the Ninth International World Wide Web Conference*, Amsterdam, The Netherlands, 13 – 15 May.
- [5] "Generating and Compiling Source Code Dynamically in Multiple Languages" (2004), MSDN Library [Online], Available: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconGeneratingCompilingSourceCodeDynamicallyInMultipleLanguages.asp>
- [6] J. Clark (ed.) (1999), "XSL Transformation (XSLT) Version 1.0 (Recommendation)", W3C, 16 November [Online], Available: <http://www.w3.org/TR/xslt>
- [7] J.I. Maletic, M.L. Collard, A. Marcus (2002), "Source Code Files as Structured Documents", *Proceedings of the Tenth International Workshop on Programming Comprehension*, IEEE Computer Society Press, Paris, France, 27 – 29 June, pp. 289 – 292.
- [8] M.L. Collard, H.H. Kagdi, J.I. Maletic (2003), "An XML-based Lightweight C++ Fact Extractor", *Eleventh IEEE International Workshop on Program Comprehension*, IEEE Computer Society Press, 10 – 11 May, pp. 134 – 143.
- [9] "System.CodeDom Namespace" (2004), MSDN Library [Online], Available: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemCodeDom.asp>
- [10] "System.CodeDom.Compiler Namespace" (2004), MSDN Library [Online], Available: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfSystemCodeDomCompiler.asp>

GOLD: A Generalized Parsing System

Devin Cook and Du Zhang

*Department of Computer Science
California State University
Sacramento, CA 95819-6021
seke@DevinCook.com, zhangd@ecs.csus.edu*

Abstract. In this paper, we describe a generalized parsing system called GOLD that can support multiple programming languages, and as a result create a consistent language development platform. The goal of the GOLD parsing system is accomplished by logically separating the component that generates parse tables for a source grammar from the component that does the actual parsing. In addition, the system supports the full Unicode character set, and has a set of language development tools that include skeleton program creation, grammar testing, and displays and exporting of parse tables. Components of GOLD have been tested on a number of source grammars. Results so far have indicated that GOLD is a useful and practical tool for programming language development. On average, approximately 3000 downloads are made per month off the GOLD website: www.devincook.com/goldparser.

Keywords: parser, parser generator, context-free grammars, compiled grammar table file, LALR and DFA tables.

1. Introduction

Currently, the most common approach to creating parsers is through the use of a compiler-compiler or parser generator. Though there are many such tools, each of them is quite different in both design and usage. As a result, developing a parser in different programming languages presents a much different experience. The grammars used by parser generators, and the features and interfaces of the tools vary greatly in both the look and the behaviour.

Since each parser generator is designed for a specific programming language, a different parser generator must be developed for each new language. Most of the common programming languages are supported by one suite or another, but newer languages and specialized languages may not have such suites.

In this paper, we describe a generalized parsing system called GOLD (Grammar Oriented Language Developer)

that can support multiple programming languages, and as a result create a consistent development environment. The goal of the GOLD parsing system is accomplished by logically separating the component that generates parse tables for a source grammar from the component that does the actual parsing. In addition, the GOLD system supports the full Unicode character set, and has a set of tools that can aid language development process. These tools include: skeleton program creations from program templates, grammar testing, display of various types of information in the parse tables, and exporting parse tables to files of different formats. The results so far have shown that GOLD is a useful and practical tool. On average, approximately 3000 downloads are made per month off the GOLD website: www.devincook.com/goldparser [6].

This paper is organized as follows. Section 2 offers some general design considerations. Sections 3 and 4 describe the Builder and the Engine component of GOLD, respectively. Section 5 discusses some implementation issues. A brief comparison is given in Section 6. Finally, Section 7 concludes the paper with remarks on future work.

2. Design Considerations

If a parser generator is to be able to support multiple programming languages, the system must not produce any information that is limited to a single language. As a result, the compiler-compiler concept [1,5] cannot be used. Instead, the parsing information created by the GOLD system should be saved to an independent file first and then later used by the actual parsing engine.

Hence, the GOLD system consists of two distinct components - the Builder and the Engine. The Builder will be the main application that analyzes a source grammar, creates parse tables and provides all other services that aid language development. The parse information created by the Builder will be subsequently saved to a file. The Engine will later load the file, read the parse tables and perform the actual runtime work.



Figure 1. Builder and Engine components in GOLD.

GOLD sanctions the following parser development cycle:

1. The grammar is defined for a programming language being developed. The description of the grammar is written using any text editor - such as Notepad or the editor that is built into the GOLD Builder.
2. Once the grammar is loaded into the system, it is analyzed by the GOLD Builder. During this process, LALR and DFA parse tables are constructed and any ambiguities or problems with the grammar are reported.
3. After the grammar is analyzed, the tables are saved to a file (called compiled grammar table file) to be used later by the actual parsing Engine.
4. The parsing Engine uses a Deterministic Finite Automata (DFA) to identify different classes of tokens and the LALR algorithm to analyze the syntax. Different versions of the Engine can be created for different programming languages and IDEs (e.g., an ActiveX DLL, or a .NET Module). Since the LALR and DFA algorithms are simple automatas, minimal coding will be necessary to implement different versions of the Engine in different programming languages.
5. The Engine reads the parse tables and analyzes a given source text. It then produces the parsing result (e.g., a parse tree) for the given source text.

One of the most important aspects of a parser generator is the format and functionality of its meta-language. The notation used by each parser generator varies greatly, and the meta-languages used by different generators are rarely compatible. To allow the easy development of source grammars, the following criteria are used in the design of the notation of the GOLD meta-language.

1. The GOLD meta-language must not contain any features which are programming language dependant. In compiler-compilers, the semantic actions are integrated directly into the meta-language. While this does aid the development of the application, it does not allow the meta-language to be used for defining other programming languages, at least without major revisions.
2. The notation of the meta-language should be very close to the standards used in language theory. This will allow both students and professionals familiar with language theory to be able to write grammars

without a large learning curve. As a result, the definitions for terminals will use regular expressions and definitions for rules will use Backus-Naur Form in the GOLD meta-language.

3. The meta-language should include all language attributes. There are many aspects of programming languages that cannot be specified using regular expressions or Backus-Naur Form. These include the actual name of the grammar, whether it is case sensitive or not, the author, etc.

3. The Builder

3.1. Meta-Language

GOLD Builder meta-language defines character sets, predefined character sets, comments, terminals, whitespace terminals, comment terminals, rules and parameters. Figures 2-6 are the grammar syntax charts for the GOLD meta-language.

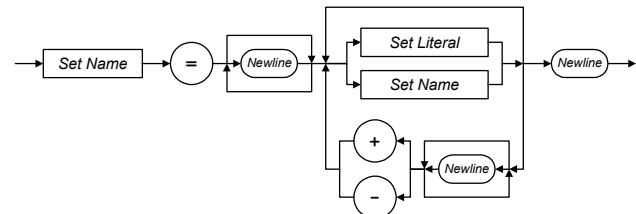


Figure 2. Set syntax.

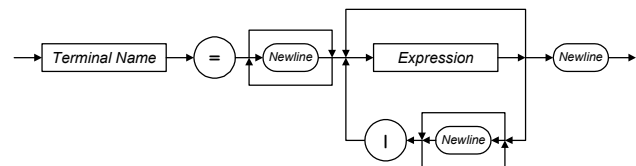


Figure 3. Terminal syntax.

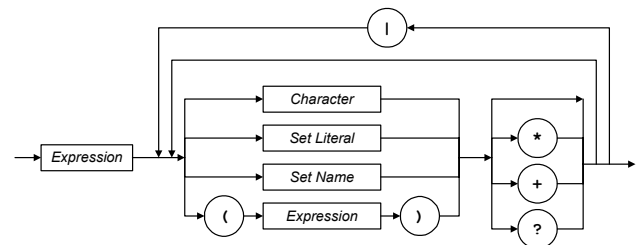


Figure 4. Regular expression syntax.

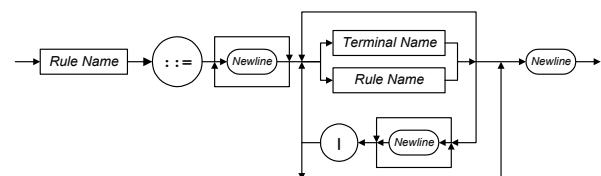
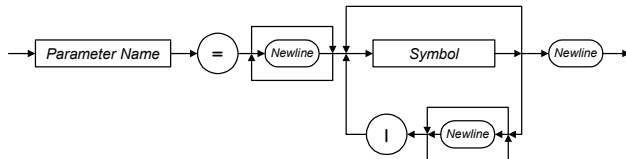


Figure 5. Rule syntax.



3.2. Compiled Grammar Table File

The compiled grammar table (.cgt) file is a file format that is designed to store the parse tables and other relevant information constructed by the Builder. The file format is defined based on the following principles:

1. The file will be written to only once when it is created. Afterwards, information will only be read sequentially from the start of the file.
2. The format should be easy to implement on numerous platforms. In other words, to be very simple structurally.
3. The file structure should be extensible, allowing data structures to be added or expanded as needed in the future. The file will store structures such as integers, Unicode strings, bytes, but other types may be necessary in future versions.
4. The file structure should allow additional types of records to be added, if needed. In the future, the file format should be able to store different types of information besides the parse tables. These includes, for instance: sound files, pictures, source code, etc.

Rather than defining a new file format, a number of existing formats were researched beforehand. The most notable of the formats is XML [19]. A natural question arises here: why not using XML?

While the Builder and the GOLD system should be friendly to the XML format, given its popularity, XML has the following drawbacks that make it not the ideal format for this project.

1. The XML format is simple, but by its nature, not very compact. The tables created by the Builder can be extensive for complex grammars. The file can be compressed by any number of algorithms. However, this would place a high degree of burden on developers, and, therefore, not acceptable.
2. XML is a simple text file format, and can be easily edited by the developer. As a result, it would be possible to hack the tables to produce results not supported by the system. To insure backwards compatibility with future versions of GOLD, developers must work within the intended system structure.

3. XML does not allow different types of files to be embedded.

The format for the compiled grammar table file is given in Figure 7 below.



Figure 7. Compiled grammar table file format.

The first data structure in the file is the file header that contains a null-terminated Unicode string. This string contains the name and version of the type of information stored in the succeeding records. Following the header, the file contains one or more records. A record in the compiled grammar table file is depicted in Figure 8.

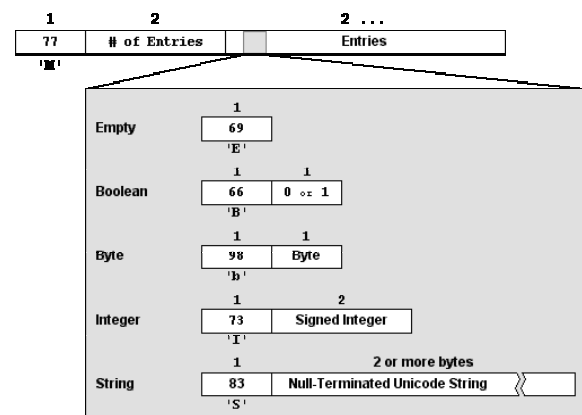


Figure 8. Compiled grammar table records.

Each record starts with a byte containing the value 77, the ASCII code for the letter "M", which stands for Multitype. It is possible, in the future, to add more types such as pictures, sounds, additional parse information, and other files. Following the first byte, there is a two-byte unsigned integer indicating the total number of entries in the record. A compiled grammar is represented using many types of entries. For instance, Figure 9 is an entry containing size information for different tables.

Byte	Integer	Integer	Integer	Integer	Integer
84	Symbol Table	Character Set Table	Rule Table	DFA Table	LALR Table

Figure 9. Table size entry.

3.3. Program Templates

Since each rule or a symbol is uniquely identify by a table index in parse tables, the Engine component must deal with rules and symbols in parse tables in terms of their table indexes. When developing a program, manually typing each constant definition can be both tedious and

problematic - given that a single incorrect constant could be difficult to debug. For most programming languages and scripting languages, the number of rules can easily exceed a hundred.

Program templates are designed to help alleviate the chores in the Engine development. A program template is a text file containing simple pre-processor type statements for a specific version (e.g., C++, Java, or Visual Basic) of the Engine. The Builder can read a program template and create the corresponding skeleton program for that version of the Engine. Depending on the needs, the skeleton program can include lists of constants, case statements, variables, and so forth.

Currently, there are program templates for a number of different programming languages. Each program template is stored in a subfolder of the GOLD Builder application.

4. The Engine

A version of the GOLD Parser Engine was developed in conjunction with the Builder for the Visual Basic 6 programming language. The code was subsequently compiled into a Microsoft ActiveX DLL and made available with the Builder. Although Visual Basic 6 has well-known limitations affecting file access and object inheritance, the language is fairly easy to read by programmers of other languages. The object hierarchy and the Visual Basic Engine interface were designed to set a simple standard that could be used as a guide for different versions of the Engine. Figure 10 indicates the Engine data flow.

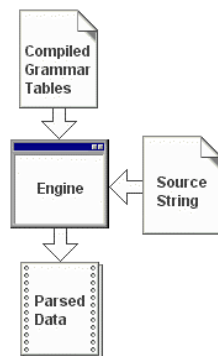


Figure 10. Engine data flow.

The source code for the Engine component was made available on the Internet and, consequently, has been converted to a number of different programming languages and development IDEs. Currently, the Engine is available in: Visual Basic .NET, Visual Basic 6, C#, C++, Visual C++, ANSI C, Delphi 5 and Java [2, 8, 10, 11, 14-18].

In essence, the Visual Basic version of the Engine is designed around a central object aptly named "GOLDParser". This object contains both the Rule Table and the Symbol Table, and performs all the parsing logic in the system including the LALR and DFA algorithms. The remaining four objects (Symbol, Rule, Reduction, and Token) are used either for storage or to support the GOLDParser object itself.

The actual parsing process is carried out through the following actions of the GOLDParser object.

1. Call LoadCompiledGrammar() method to load a compiled grammar table file.
2. Call the appropriate method to open the source string to be parsed.
3. Continue to call the Parse() method until the string is either accepted or an error occurs.

5. Implementation

The main component of the GOLD system is the Builder, which currently runs on the Windows 32-bit operating systems (Windows 9x, Windows NT and Windows XP). In addition to the main function of reading a source grammar written in the GOLD meta-language, generating the LALR and DFA parse tables, and saving the information to a compiled grammar table file, the Builder also has the following set of features that aids the language development process:

1. It can create skeleton programs using the program templates.
2. It supports the interactive testing of grammars through its integrated version of Visual Basic Engine.
3. It contains a simple text editor for source grammar editing purpose.
4. It allows many aspects of a source grammar to be displayed through different windows (parameters, symbol table, rule table, log information, DFA state table, and LALR state table).
5. It can export a source grammar's information and computed tables to a web page, a formatted text, or an XML file.

Figure 11 is an annotated screenshot of the GOLD GUI.

The flow of the application is broken down into three distinct steps that progress from a source grammar to the completed parse tables. The developer advances by clicking the "Next" button that is located on the bottom left side of the main window.

1. Enter the source grammar. During this step, the Builder checks the syntax of the grammar itself and prepares the system to compute the parse tables. If

the grammar contains an error, it is reported to the user and the system resets.

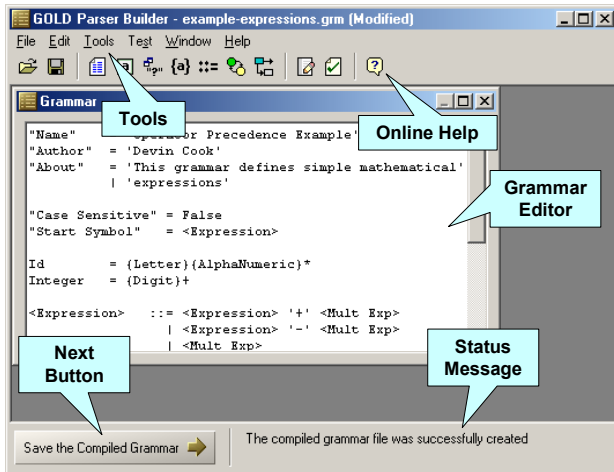


Figure 11. GOLD Builder GUI layout.

2. Compute the parse tables. During this step, the Builder attempts to construct the LALR and DFA parse tables. Most conflicts occur during the construction of the LALR parse tables, as a result, these are constructed first.
3. Save to the compiled grammar table file. If no problems are found, the developer now has the ability to save the parse tables to a compiled grammar table file. If the developer clicks on the button at this point, the system will automatically display the "Save File" dialog window. Since the parse tables are complete and ready to use, the developer also has the ability to interactively test the grammar, export the grammar to different file formats, and create skeleton programs.

If a terminal is not functioning correctly, the developer can review the actual DFA used by the Engine's tokenizer. This window also allows students to view the actual information produced by the GOLD Parser Builder (Figure 12).

State	Action(s)	Characters
0	Goto 1 Goto 2 Goto 3 Goto 4 Goto 5 Goto 6 Goto 7 Goto 8 Goto 9	{HT} {LF} {VT} {FF} {CR} {Space} {NBSP} { } * + - / 0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz...
1	Goto 1 Accept (...)	{HT} {LF} {VT} {FF} {CR} {Space} {NBSP}
2	Accept '{'	

Figure 12. DFA states.

Most errors that occur in grammars are found in the LALR State Table. When the system analyzes a grammar and computes the parser tables, often shift-reduce and reduce-reduce conflicts are found.

The LALR State Table Window (Figure 13) allows the developer to review the produced states – in particular the state that contains the error.

State	Configuration/Action	Lookahead
	Integer Shift 4 <Expression> Goto 19 <Mult Exp> Goto 17 <Negate Exp> Goto 13 <Value> Goto 10	
1	<Negate Exp> ::= '-' * ... <Value> ::= * Id <Value> ::= * Integer <Value> ::= * '(' <Expr... '(' Shift 2 Id Shift 3 Integer Shift 4 <Value> Goto 18	EOF -) * / + EOF -) * / + EOF -) * / + EOF -) * / +
2	<Value> ::= '(' * <Expr... <Expression> ::= * <Exp...	EOF -) * / + -) +

Figure 13. LALR states.

One of the most important tools in the GOLD Builder is the Grammar Test Window. After the Builder has successfully compiled a source grammar, the developer can use this tool to test the grammar with regard to test cases. There are three separate "tabs" in the Grammar Test Window. The "Source" tab allows the developer to enter or load a test case. After the test case is ready, the developer can click the "Parse Actions" tab to start the parsing process for the test case. If the test string is successfully parsed, by clicking "Parse Tree" tab, a parse tree is produced for the user to review. This tree can be saved to a formatted text. Figure 14 shows a parse tree for the test case: "a+b*c-d".

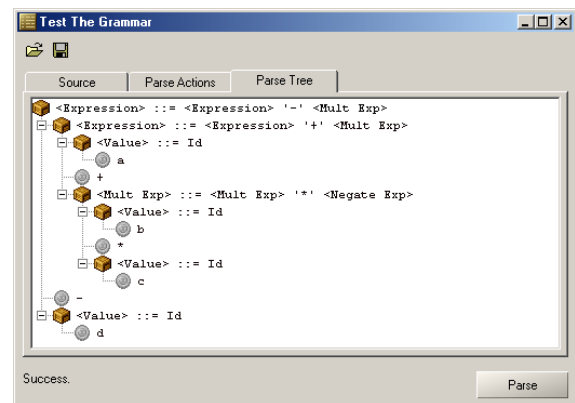


Figure 14. Grammar test window.

In addition to some small source grammars, GOLD has been tested on source grammars for BASIC, ANSI C, COBOL, HTML, LISP, Smalltalk, SQL, Visual Basic .NET, XML and GOLD meta-grammar.

6. Comparison

Currently, the field of computer programming languages has witnessed a myriad of different parser generators with different features, interfaces and meta-grammars [7]. Due to space limitation, we compare GOLD with some of the existing parsing tools.

Yacc is one of the oldest parser generators [9]. For developers using C or C++ on the UNIX platform, Yacc is an ideal tool. It has the advantage of close integration between the source code and the special directives. Its limitations are that the meta-grammar definition is not portable between systems, and that there is lack of support for component based design and object-oriented programming.

ANTLR is an object-oriented parser generator that is capable of generating parsers for several programming languages which have the same basic syntax as C++ (Java, C#, and C++) [13]. It has its meta-language that can be used to define the names and properties of different classes to be generated. Each class inherits one of the three built-in ANTLR classes: Parser, Lexer, and TreeParser, and uses the LL(k) parsing algorithm.

Bison is a general-purpose parser generator that converts a grammar description for an LALR context-free grammar into a C program to parse that grammar [4]. Bison is upward compatible with Yacc: all properly-written Yacc grammars ought to work with Bison with no change.

Elkhound is a parser generator based on the generalized LR (GLR) parsing algorithm [12]. Because of GLR, Elkhound can parse with any context-free grammar, including those that are ambiguous or require unbounded lookahead. Due to an improvement to the GLR algorithm, Elkhound parsers can achieve a performance that is as fast as LALR(1) parsers on the deterministic portions of the input.

GENOA is a framework for code analysis tools for software engineering tasks [3]. Its front end involves parsing, and a language-independent abstract syntax tree (AST), to which the .cgt file in GOLD is akin.

Compared with the aforementioned tools, GOLD has the following benefits: It supports multiple programming languages and the full Unicode character set. It has a set of development tools. Its meta-language is easy to understand, and its Builder GUI is easy to use in programming language development.

7. Conclusion and Future Work

Currently, the Builder is only available on the Windows 32-bit platforms. Although this makes it accessible to a wide number of students and computer scientists, different versions are needed for Linux, UNIX and a command-line version for Windows.

The source code for the Builder is written in Visual Basic 6, and will have to be translated to C++ for this process.

Also, additional enhancements to the meta-language syntax and semantics are needed to handle grammars that are not fully context-free. Python, for instance, cannot be parsed by a pure context-free parsing system.

References

- [1] A. W. Appel, *Modern Compiler Implementation in C*. Cambridge University Press, 40 West 20th Street, New York City, New York 10011-4211, 1998.
- [2] M. Astudillo, C++ GOLD Parser Engine. e-mail: d00mas@efd.lth.se.
- [3] P.T. Devanbu, "GENOA – A Customizable, Front-end Retargetable Source Code Analysis framework", *ACM Transactions on Software Engineering and Methodology*, Vol.8, Issue 2, April 1999, pp.177-212.
- [4] C. Donnelly and R.M. Stallman, "Bison: the Yacc-compatible Parser Generator (Bison version 1.35)", Free Software Foundation, 675 Mass Ave, Cambridge, MA February, 2002.
- [5] C.N. Fischer and R.J. LeBlanc, *Crafting A Compiler*, Benjamin/Cummings Publishing Company, 1988.
- [6] GOLDParse, <http://www.devincook.com/goldparser>.
- [7] Google's directory of Lexer and Parser Generators, http://directory.google.com/Top/Computers/Programming/Compilers/Lexer_andParser_Generators/.
- [8] M. Hawkins, Java GOLD Parser Engine. <http://www.hawkini.com>.
- [9] S. C. Johnson, Yacc: "Yet Another Compiler-Compiler", TR 32, AT&T Bell Labs, 1975.
- [10] I. Khachab, Modified Delphi GOLD Parser Engine. e-mail: ibrahim@euronia.it.
- [11] M. Klimstra, C# GOLD Parser Engine. e-mail: klimstra@home.nl.
- [12] S. McPeak, "Elkhound: A Fast, Practical GLR Parser Generator", TR No. UCB/CSD-2-1214, December 2002.
- [13] T. Parr, ANTLR Website, <http://www.antlr.org>.
- [14] A. Rai, Delphi GOLD Parser Engine. e-mail: riccio@gmx.at.
- [15] E. Ugurel, C++ GOLD Parser Engine. e-mail: eylemugurel@hotmail.com.
- [16] M. van der Geer, Delphi GOLD Parser Engine. email: Beany@cloud.demon.nl.
- [17] R. van Loenhout, C# GOLD Parser Engine. e-mail: rvl@software-engineer.net.
- [18] R. Wilbanks, Visual Basic .NET GOLD Parser Engine. e-mail: rwilbanks@starband.net.
- [19] World Wide Web Consortium, *Extensible Markup Language (XML)*, <http://www.w3.org/XML/>.

Grammatically Interpreting Feature Compositions

Wei Zhao¹, Barrett R. Bryant¹, Fei Cao¹, Rajeev R. Raje², Mikhail Auguston³,
Carol C. Burt¹, and Andrew M. Olson²

¹ *Computer and Information Sciences, University of Alabama at Birmingham,
Birmingham, AL 35294-1170, USA. {zhaow, bryant, caof, cburt}@cis.uab.edu*

² *Computer and Information Science, Indiana University Purdue University Indianapolis,
Indianapolis, IN 46202, USA. {rraje, aolson}@cs.iupui.edu*

³ *Computer Science, Naval Postgraduate School, Monterey, CA 93943, USA
auguston@cs.nps.navy.mil*

Abstract. Feature modeling is a popular domain analysis method for describing the commonality and variability among the domain products. The current formalisms of feature modelling do not have enough support for automated domain product configuration and validation. We have developed a theory of feature modeling: a feature model is analogous to a definition of a language; a particular feature composition instance (domain product) is analogous to a program written in that language; and the way the features can be assembled to form a product is analogous to the way various tokens can be assembled to form a program. To apply this theory, we have developed a meta-language Two-Level Grammar++ to specify feature models. The interpreter derived from the feature model specification performs automated product configuration and product quality validation.

1. Introduction

The systematic discovery and exploitation of commonality across related software systems is a fundamental technical requirement for achieving successful software reuse [13]. Domain analysis is one technique that can be applied to meet this requirement. Feature modeling is a popular domain analysis method originated in [11]. The current formalisms [6], [11] of feature modeling do not have enough integrity for supporting automated domain product configuration and validation (some tools were implemented to support limited automation up to the power of the original formalism, e.g. [4]). We have developed a theory of feature modeling so that the existing compiler technologies can be leveraged for automated domain product configuration.

A feature is a distinguishable characteristic of a concept that is relevant to the stakeholder of that domain [6]. We have defined that the anatomy of a feature is a modular encapsulation of three-dimensional views: an

abstract view at the domain business level, a constructive view at the architectural pattern level and a concrete view at the implementation technology level. The artifacts in this encapsulation consist of both code and non-code. Examples of the artifacts are business domain models, design models, make files, HTML documents, XML descriptors, etc. We consider features to be concrete and non-cross-cutting concepts of a domain, i.e., a feature can be incarnated as a software component with specific programming and component technologies.

We consider a feature model to be a general specification of a domain: the rules about feature configurability and how to manufacture the valid product instances in that domain. So, a feature model is a definition of feature compositions. By observing that any language (machine, assembly, and high level languages) is a composition of language elements (constructs and tokens) at different abstraction levels, we are motivated to develop a language-based theory of feature modeling: a feature model is analogous to a definition of a language; a particular feature composition is analogous to a program written in that language; the way the features can be assembled to form a product is analogous to the way various tokens can be assembled to form a program; the interdependency relationships among the feature models are analogous to the object relationships that can be defined in object-oriented programming languages. A valid product for a domain can be created by composing a set of features adhering to the composition rules in the feature model. In a feature model, there are atomic features and composite features. An atomic feature is a feature that does not need to be further refined into sub features when there are no variations among different products. A composite feature is a composition of one or more atomic or composite features. Both the atomic and composite features are relative concepts. A composite feature in one feature model can act as an atomic feature in a foreign feature model. This hierarchy is called the

feature organization, and the structure of a product is called the product organization.

To apply successfully the programming language techniques to feature modeling, the first question to be answered is whether there exist concepts in feature models that are the counterparts of syntax and semantics in programming languages. The syntax of the feature model is the business domain feature organizational structure. The static semantics indicates the configuration constraints such as feature attributes, feature relationship cardinalities, interdependencies, and domain-specific business operational rules. The dynamic semantics models the states of system property changes after the steps of feature compositions. That includes pre- and post-conditions for the configurations, temporal concerns, and the Quality of Service (QoS) attributes [14], [17]. An example of a QoS attribute is transaction speed in the banking domain. We draw a clear delineation of semantics of a composition model (feature model) from semantics of a composed system. The semantics of a feature model is the non-functional quality aspect of a composition; the semantics of the composed system is the functional quality aspect of a composition. Feature model syntax defines the semantics of the composed system meaning that as long as the features are composed in a proper hierarchy, the composed system should function correctly assuming correct feature implementations and correct feature model. For example, if we build a money transfer system by composing features withdraw and deposit, the balance calculation is the semantics of the composed system, whereas the transaction speed calculation is the semantics of the composition model.

We have developed a meta-language called Two-Level Grammar++ (TLG++), an object-oriented extension of Two-Level Grammar (TLG) [18], to specify feature models. TLG, a Turing complete grammar, has been used for integrated definition of programming language syntax, static semantics and dynamic semantics, which makes TLG ideal for specifying the feature organization along with static configuration constraints and various dynamic semantic concerns. Because of object-oriented features, TLG++ naturally fits in the conceptual modeling of inter-connected object relationships among the feature organizations. The interpreter derived from the feature model specification performs automated product configuration and predicted product functional and non-functional quality validation.

According to the three-dimensional views of domain features, there are three dimensions of feature compositions: semantic-business composition, syntactic-architecture composition, and lexical-technology composition. For a particular product created by composing a set of features, the semantic-composition

dimension defines the entangled business logic among the features and semantics for individual features; the syntactic-composition dimension defines a compositional architecture for this product; the lexical-composition dimension defines how each feature is technologically formed thus contributing to the binary connection, interoperation and deployment between any two feature-implementations. In this paper we only demonstrate the first dimension. However, a complete product quality validation requires all three-dimensional composition validations.

The following section introduces TLG and TLG++. A case study is given in section 3. Section 4 compares our work to related work. The paper concludes in section 5.

2. Two-Level Grammar++

Two-Level Grammar (van Wijngaarden grammar or W-grammar) is an extension of Context-Free Grammar (CFG) and was originally developed to define syntax and semantics of programming languages. It has been shown that TLG defines the family of recursively enumerable sets [15], and suitable restrictions yield context-sensitive languages [1]. It has been used to define the complete syntax and static semantics of Algol 68 [18] and dynamic semantics of programming languages [5]. Recently, it was developed as an object-oriented requirements specification language integrated with VDM¹ tools for UML² modeling and Java/C++ code generation [3].

The term “two-level” comes from the fact that a TLG is composed by two finite sets of CFG rules: a set of formal parameters may be defined using a CFG, with the possible generated strings used as arguments in predicate functions defined using another CFG. Originally, the first level CFG rules were called the meta-productions/rules, while the second level parameterized CFG rules were called hyper-productions/rules. After the meta-rules get substituted into the hyper-rules, a third implicit and possibly infinite set of CFG rules, called the production-rules, are derived. It is the production-rules that finally generate the target language that a TLG describes.

TLG++ is the object-oriented TLG for specifying feature models. Moving from TLG to TLG++ poses a paradigm shift. TLG is used to physically define programming language syntax and semantics, while TLG++ is used to abstractly define the concepts of a domain. To specify the feature model, we do not have the concept “terminal symbol in the target language” in TLG++. In fact, the terminals in the feature models are either the atomic features or domain-specific keywords.

¹ VDM – Vienna Development Method – <http://www.ifad.dk/vdmtools>

² UML – Unified Modeling Language – <http://www.omg.org/uml>

Examples of “domain-specific keyword” might be: a particular domain logic control pattern, domain-specific algorithm, and so on.

The atomic features in a feature model are represented by Universal Resource Identifiers (URIs)³. In the process of interpreter generation, the URIs are simply treated as terminals. While interpreting a specific product, there are two cases under consideration: 1) the atomic feature in this feature model is a composite feature in another feature model and there is no direct implementation for this atomic feature, then preprocessing can be adopted to ensure the instance atomic feature used in this particular product is in fact a valid instance defined by the corresponding URI, which might just involve another interpretation process; 2) this atomic feature has a direct implementation, in which case the Unified Meta-component Model (UMM)⁴ needs to match the URI to complete the interpretation. From a single feature model perspective, the URI is treated programmatically rather than syntactically or semantically. Reasons for the use of URIs are:

First, the URI for an atomic feature or the URI compositions for a composite feature are the types of the component that implements this feature. Because of the nature of composition, the atomic feature has a single type, and the composite feature has type variations.

for example, $A :: B \ C; \ D. \ B :: E; \ F.$ (5) in (5), A and B are composite features, and C, D, E, and F are atomic features. “;” means “or”. Suppose the URI for each atomic feature is the `http://` plus the letter symbol. So, atomic feature C has type `http://c`, D has type `http://d`, and so forth. The types of a composite feature are the composition of types of atomic features. The composition is a tree structure. The number of types of a composite feature equals the number of strings it can generate. In this case, A has 3 types shown in fig. 1. If the component developer chooses to implement a composite feature directly, he/she must identify a specific type of choice. A component is considered plug-compatible for another component that implements the same feature if and only if their types match. One reason we developed TLG++ to specify the feature model is because TLG++ naturally supports this hierarchical type structure since each parameter is defined by a context-free grammar.

Second, the use of URI gives the potential to specify very large and highly distributed domains, as some fairly complex features can be defined separately and linked by a URI. The mechanism of URI complements object-orientation for distribution and encapsulation.

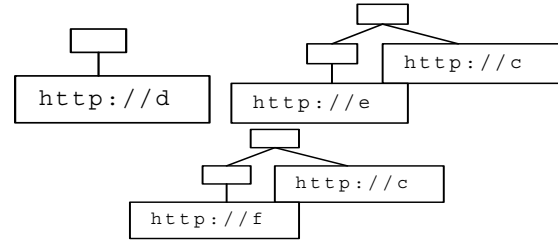


Fig. 1. The feature A has three types.

Third, compositions can be easily reused, e.g., a composite feature represented by a URI in a domain can stand as an atomic feature in another domain.

Lastly, the separation of the feature id from the feature representation allows features to have any physical form that the designer wishes for designing the visualized domain analysis tools, such as an icon, a name, or a simple box.

Object-orientation is proper for modeling real-world relationships. The feature models specified in TLG++ naturally model object relationship graphs in real-world business domains, since both domains of different categories and domains (sub-domains) in different levels of the hierarchy of the same category can have a feature model. Real-world domains are usually hierarchical: one narrower-scoped domain inherits the concepts from many other broader-scoped domains. With the encapsulation, a feature model for a domain is expressed in a main class, from which other classes might be linked. TLG++ has a root class `Notion` that is comparable to the `Object` class in the Java programming language. The class `Notion` groups general notational extensions (e.g., list, sequence and tree definition) [5], built-in data types, and primitive grammatical computations. Those pre-established grammatical concepts are inherited by any other TLG++ classes. Inheritance and encapsulation provide TLG++ much more power than CFG in terms of reusability and modularity. By polymorphism, users do not need to cite the `Notion` class in order to call its rules. A class can override the rules in its parent classes, and any class can override the rules in the `Notion` class.

3. A Case Study

In this section, we give an example on how a feature model is formulated, how a feature model is represented in TLG++, and how a product can be validated based on the feature model.

Fig. 2 shows a fragment of the feature model for a `PersonalAccount` domain. From the viewpoint of stakeholders of this domain, the feature model should capture the distinguished domain concepts (i.e., features: objects and operations) and the business rules on how

³Naming and Addressing, <http://www.w3.org/Addressing/>

⁴ UMM is the meta-model for feature implementation. Detailed explanation of UMM can be found in [13].

those concepts are composed to form a product. For the sake of this example, we assume the PersonalAccount domain has an object (PersonalAccount), and the operations (Withdraw, Deposit, and MoneyTransfer). MoneyTransfer is a composite feature composed of Withdraw and Deposit.

```

class PersonalAccount extends Banking.
  Account :: Integer. 1
  Customer :: Name SocialNumber. 2
  Bank :: Integer. 3
  Balance :: Integer. 4
  Amount :: Integer. 5
  TurnAround :: Integer. 6
  Name :: String. 7
  SocialNumber :: Integer. 8
  PersonalAccount::http://omg.org/bankdoma
    in/personalAccount. 9
  Withdraw ::
    http://omg.org/bankdomain/withdraw. 10
  Deposit ::
    http://omg.org/bankdomain/deposit. 11
  MoneyTransfer :: . 12
  ....
  turnaround TurnAround1 moneyTransfer
  MoneyTransfer : 13
    turnaround TurnAround2 customer
    Customer withdraw amount Amount1 from
    Withdraw,
    personal account customer Customer has
    account Account1 in bank Bank1 with
    balance Balance1 PersonalAccount,
    turnaround TurnAround3 customer
    Customer deposit amount Amount2 to
    Deposit,
    personal account customer Customer has
    account Account2 in bank Bank2 with
    balance Balance2 PersonalAccount,
    where Balance1 != 0,
    where Amount1 = Amount2,
    where Account1 != Account2,
    where TurnAround1 = TurnAround2 +
      TurnAround3.
  ....
end class.

```

Fig. 2. Bank domain feature model in TLG++

In the TLG++ representation, the first thing to be noticed is the separation of meta-rules and hyper-rules. Rules 1 to 12 are meta-rules, and rule 13 is a hyper-rule differentiated by using “::” and “:”. Parameters start with a capitalized letter. The values (generated strings) of parameters defined in the meta-rules are called the Terminal Meta-Production (TMP) of parameters. Plugging the TMPs into the hyper-rules, we get the production rules. This parameterization, called the Uniform Replacement Rule (URR), is the essential theory that distinguishes the TLG from the pure CFG. The rule is that each occurrence of a parameter in a single hyper-rule needs to be replaced by the same TMP of that parameter.

A parameter followed by a number is a new distinct parameter with the same definition as the root parameter. In the rule 13 Account1 and Account2 will not necessarily be replaced by a same TMP of Account.

The convention we used is: the meta-rules are used to define the hierarchical context-free type structure for parameters; and the hyper rules define the syntax and semantics for feature compositions. Integer and String are built-in data types defined in the Notion class. Rule 12 is an empty definition, which shows that MoneyTransfer is a new composite feature to be defined in hyper rules. We assume the domain specifications should be created by domain experts working with standards organizations such as OMG⁵.

Rule 13 specifies the syntax and semantics for the MoneyTransfer composition. TLG++ rules are natural language based, and are flexible in terms of writing styles. The convention we have adopted is that some words indicating the meaning are followed by the parameter, e.g., turnaround TurnAround1. In rule 13, the atomic feature is expanded with its specific parameters, e.g., turnaround TurnAround2 customer Customer withdraw amount Amount1 from stands for atomic feature Withdraw. Each rule begins with the syntax definition followed by the definition of semantics in where clauses. The static semantics here includes: the customers must be identical (ensured by URR); the accounts must be distinct; the amount withdrawn and deposited must be the same; the account to be withdrawn from must have a positive balance. Recall that the dynamic semantics of the feature composition and the dynamic semantics of the composed system are distinct. The balance calculations, after the actions withdraw and deposit, are the dynamic semantics of the composed system, which we will not be able to specify in a feature model. One QoS parameter, TurnAround time, is defined as the composition dynamic semantics. This example should convince the reader that features until being implemented as components are static concept entities rather than computation entities.

PersonalAccount is a sub-class of Banking where some basic features and feature compositions can be inherited. Polymorphism may exist as well. For example, the syntax definition of MoneyTransfer could be moved up to the Banking feature model. There might be another class BusinessAccount sub-classing Banking. So, the BusinessAccount feature model only needs to specify its specific semantic rules such as the requirement of a special security monitor for the MoneyTransfer composition, and TurnAround <=10.

Suppose the product we are trying to build is a simple MoneyTransfer system that can be created by

⁵ OMG - Object Management Group - <http://www.omg.org>.

composing Withdraw and Deposit. For the validation of this product, the following are important points:

1. The goal of the validation is to find out if the feature compositions (the business logic or semantics of the product) are correct, and whether the product will have expected QoS using the supplied components.
2. The composition of components in this example occurs dynamically, so the state is an important concept. The state of a running component that implements a particular feature is the business data currently maintained. Please note that this paper presents the composition in the semantic-business dimension, not on the architecture or implementation dimension, so the state is not the state of the machine that runs this component. There are two cases regarding the state: first, the component is already running; second, the component has been produced, but is not yet running. In the second case, the state is the initial state, i.e., the state that is instantly after the component is invoked. In both

cases, the component is treated as a black box. Those two cases give the views of dynamic product-line assembly and static product-line assembly respectively. The example in this section is of the first case.

3. Yet the validation for the composition is static. We are not going to run the system in order to test if the system is built by a correct composition. The UMMs of the implementation components provide the feature URIs, the QoS values, and the states information. This information comprises a sentence that stands for the product we are going to build; and this sentence should be interpreted according to the feature model definition. Currently, the UMM is represented in XML and is generated automatically by the tool support from the component developer [14]. We are investigating how to convert the XML based representation to a string of text so that the product can be interpreted, or to extend the ability of the interpreter to interpret the XML strings directly.

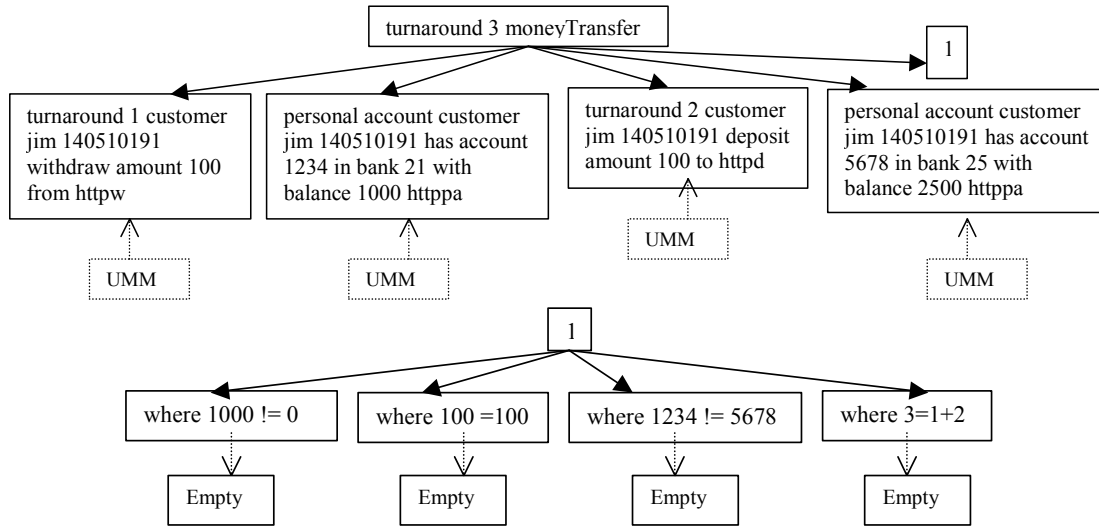


Fig. 3. The parse tree for the product MoneyTransfer. httpw, httppa, httpd stand for the URIs of Withdraw, PersonalAccount, and Deposit, respectively.

Fig. 3 provides a parse tree for a sample MoneyTransfer product to show how TLG++ can grammatically interpret this particular product to validate both functional and non-functional composition. As this composition is simple, the parse tree is not so deep. The QoS attribute TurnAround time for this product is expected to be 3 milliseconds. The state information and the QoS value for the implementation components are randomly selected for the illustration of this example. For an easier understanding, the interpretation process illustrated in fig. 3 is top-down and directly depends on the implicit production-rules, i.e., all the parameters have been non-deterministically substituted into the hyper-rules

before the interpretation process begins. In fig. 3, consider when we interpret the product turnaround 3 moneyTransfer, we apply rule 13 because the parameter TurnAround1 has been non-deterministically and implicitly substituted by its value 3 before the interpretation begins.

From this bank domain feature model specification, the bank domain product interpreter can be generated automatically using our tool—the TLG++ interpreter. The TLG++ interpreter uses the CUP⁶ parser generator once

⁶ CUP – Construction of Useful Parsers - <http://www.cs.princeton.edu/~appel/modern/java/CUP>

for the meta-rules and once for the hyper-rules to generate two sets of parsers. So, from the implementation point of view, the interpretation process of a product is bottom up and driven by the hyper-rules looking up the generated parser for each parameter whenever it encounters a parameter in the hyper-rules. This look-up process resembles looking-up the value of variables in the symbol table during the interpretation of programming languages. For example, while parsing the money transfer product, in order to apply rule 13, the parser for the parameter TurnAround is picked up and parses the string 3 to test if it may be derived by TurnAround.

4. Related Work

Feature Diagrams. In the literature, a feature model usually includes [6]: a feature diagram that portrays feature organization; feature semantic definitions; feature composition rules and configuration constraints; rationale for features indicating the reason for choosing or not choosing a given feature. Normally, the feature diagram is represented graphically by a CASE tool, and other semantic aspects of the feature model are annotated using natural language [12], or are linked to other more formal techniques such as object diagrams, interaction diagrams, state-transition diagrams, and synchronization constraints [6]. The separation of the feature diagram and its semantic aspects drastically hinders the automated configuration and validation of domain products. Furthermore, the popular feature diagram computation model is rather primitive in terms of computation power of the mathematical computation models. When the layers of the feature diagram are flattened, the terminal productions (a set of terminal features generated from the feature diagram) can be represented by a regular expression. The tree-shaped feature diagram is even less powerful than the regular expression because the star operation in regular expressions does not have a counterpart in the feature diagram. Compared to the feature diagram, TLG++ is much more powerful in computation and presents better integrity in representing both the syntactic and semantic aspects of feature models.

Domain specific languages. Domain-Specific Languages (DSLs) [7] always have the pre-constructed notations and abstractions offering expressive power for a particular domain. No matter whether a DSL is in a graphic form, or in textual form, it has its own syntax and semantics definition. But in this paper, we define the domain directly as a language. Any compositions in the domain are the relationships presented by the grammar rules and are not physically represented by any non-grammatical symbols (+, *, etc.), or built-in operator notations in the meta-language. We call this an *open*

operator definition, which gives much flexibility to the meta-language for the evolution of operators of a domain, i.e., we only need add some new TLG++ rules for the new operators.

Composition Language. Composition Language (CL) [8] has defined composition semantics (QoS) such as latency, safety, and availability on the component model level. It did not address how to formalize QoS in the dimension of business domain semantics.

GenVoca. GenVoca is a software system generator [2]. The composition validation in GenVoca is also based on the claim that the domain defines a grammar whose sentences are software systems. Attribute grammars are used for the design rules validation including pre/post condition and pre/post restrictions. Although the model of validation sounds similar, there are some fundamental differences. In GenVoca, the principle for component composition is parameterization among components, and hence the composition is directly coupled with the component implementation language. In this paper, the composition is defined by the domain feature organizational structure and the associated semantic rules. This give a higher level of view of composition and the features can be potentially implemented in different technologies.

5. Conclusions

We have offered a foundation for the feature composition and an automated way to validate a composed system. We have addressed how both the functional and non-functional aspects of composition can be formally modeled and validated.

The method chosen to specify the feature model is Two-Level Grammar++. We could choose attribute grammar to specify the feature model, and it also provides Turing computability. However, just as the attribute grammar is not proper for specifying programming language dynamic semantics [9], it is not proper for specifying the dynamic semantics of feature composition. Compared to a regular programming language or other formal notations such as Z [16] and feature logic [20], a grammar has better constructs and computation mechanism for specifying languages. Specifically, TLG++ as a meta-language chosen in this paper has the following advantages:

1. The ability of TLG++ to integrate the feature model syntax and semantics into one formal grammar notation gives formal consistency and completeness of the specification, and eliminates the task of building a separate interpreter. Compared to the conventional way that defines the syntax with a grammar and explains the

semantics in natural language, the integration of syntax and semantics poses easier maintenance.

2. TLG has a context-free hierarchical type structure, which supports the type system in feature modeling.
3. Because both the meta and hyper rules of TLG++ are CFGs, the derivation of the product parser/interpreter can be automated and facilitated by existing parser generators, which also makes the implementation of the TLG++ meta-language easier.
4. As can be seen from the examples, the natural language style of TLG++ rules improves the language flexibility and readability.

TLG++ is simple (the only rule is URR) and flexible (natural language based). Flexibility presents a great descriptive potential but also gives the disadvantage to well control the language. The notation is not desirable to be directly used by the domain engineers, so we are investigating embedding the grammatical interpretation engine into a domain specific modeling tool such as GME [10], [19] to complement the graphic modeling notations with the automated semantic interpretation. Furthermore, formally specifying the feature models are magnitudes harder than specifying programming languages because in the programming language domain, common patterns of language constructs are well known and the conventions of writing a correct and complete specification are easy to establish. We have experienced that not only the formulation of a domain abstraction, but also the establishment of TLG++ conventions for the purpose of feature model specification are inventive and challenging tasks.

6. Acknowledgement

This research is supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

References

- [1] J. L. Baker, Some Formal Properties of the Syntax of ALGOL 68, Doctoral Dissertation, University of Washington, 1970.
- [2] D. Batory, B. J. Geraci, "Composition Validation and Subjectivity in GenVoca Generators", IEEE Trans. Softw. Eng., Vol. 23, No. 2, pp. 67-82, 1997.
- [3] B. R. Bryant, B.-S. Lee, "Two-Level Grammar as an Object-Oriented Requirements Specification Language," Proc. 35th Hawaii Int. Conf. System Sciences, Vol. 9, 2002.
- [4] F. Cao, Z. Huang, B. Bryant, C. Burt, R. Raje, A. Olson, M. Auguston, "Automating Feature-Oriented Domain Analysis," Proc. of the 2003 International Conference on Software Engineering Research and Practice (SERP'03), CSREA Press, pp. 944-949, 2003.
- [5] J. C. Cleaveland, R. C. Uzgalis, Grammars for Programming Languages, Elsevier North-Holland, Inc., 1977.
- [6] K. Czarnecki, U. W. Eisenecker, Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000.
- [7] A. van Deursen, P. Klint, J. Visser, "Domain-Specific Languages: An Annotated Bibliography", CWI, 2000, <http://homepages.cwi.nl/~arie/papers/dslbib/>
- [8] J. Ivers, N. Sinha, K. Wallnau, "A Basis for Composition Language CL", Technical Note, CMU/SEI-2002-TN-026, 2002.
- [9] G. E. Kaiser, "Incremental Dynamic Semantics for Language-based Programming Environments", ACM Trans. Program. Lang. Syst. Vol. 11, pp. 169-193, 1989.
- [10] GME User's Manual. The Institute for Software Integrated Systems, Vanderbilt University. <http://www.isis.vanderbilt.edu/projects/gme/>
- [11] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report, CMU/SEI-90-TR-21, 1990.
- [12] K. Lee, K. C. Kang, J. Lee, "Concepts and Guidelines of Feature Modeling", Proc. 7th Int. Conf. Software Reuse, pp. 62-77, 2002.
- [13] R. Prieto-Diaz, "Domain Analysis: An Introduction", ACM SIGSOFT Softw. Eng. Notes Vol. 15, pp. 47-54, 1990.
- [14] R. R. Raje, M. Auguston, B. R. Bryant, A. M. Olson, C. C. Burt, "A Quality of Service-Based Framework for Creating Distributed Heterogeneous Software Components," Concurrency Comput.: Pract. Exp. Vol. 14, pp. 1009-1034, 2002.
- [15] M. Sintzoff, "Existence of van Wijngaarden's Syntax for Every Recursively Enumerable Set," Ann. Soc. Sci. Bruxelles, Vol. 2, pp. 115-118, 1967.
- [16] J. M. Spivey, The Z Notation: A Reference Manual. Prentice Hall, New York, 1989.
- [17] C. Sun, R. Raje, A. Olson, B. Bryant, M. Auguston, C. Burt, Z. Huang, "Composition and Decomposition of Quality of Service Parameters," Proc. 5th Int. Conf. Algorithms and Architectures for Parallel Processing, pp. 273-277, 2002.
- [18] A. van Wijngaarden, "Revised Report on the Algorithmic Language ALGOL 68," Acta Informatica, Vol. 5, pp. 1-236, 1974.
- [19] W. Zhao, B. Bryant, J. Gray, C. Burt, R. Raje, M. Auguston, A. Olson. "A Generative and Model Driven Framework for Automated Software Product Generation". Proc. of the 6th ICSE Workshop of Component Based Software Engineering, pp. 103-108, 2003
- [20] A. Zeller, G. Snelting, "Unified Versioning Through Feature Logic", ACM Transactions on Software Engineering and Methodologies, Vol. 6, No. 4, pp. 398-441, 1997.

INFORMATION INTEGRATION ARCHITECTURE DEVELOPMENT: A MULTI-AGENT APPROACH

Stéphane Faulkner, Manuel Kolp, Tai Nguyen, Adrien Coyette, Tung Do
*Information Systems Research Unit, University of Louvain,
1 Place des Doyens, 1348 Louvain-la-Neuve, Belgium
Email: {faulkner, kolp, nguyen, coyette, do}@isys.ucl.ac.be*

Abstract. Multi-Agent Systems (MAS) architectures are gaining popularity for building open, distributed, and evolving software required by systems such as information integration applications. Unfortunately, despite considerable work in software architecture during the last decade, few research efforts have aimed at truly defining patterns and languages for designing such multi-agent architectures. We propose a modern approach based on organizational structures and architectural description languages to define and specify multi-agent architectures notably in the case of information integration system design as illustrated in this paper.

1 INTRODUCTION

Architectures for integrating information extracted from multiple heterogeneous sources allow to effectively exploit the numerous sources available on-line through the World Wide Web. Such architectures permit users to access and query numerous information sources to obtain an integrated answer. The sources may be conventional databases or other types of information, such as collections of Web pages.

Designing information integration systems can rapidly become complex. Indeed, such processes require software architecture to operate within distributed environments that must evolve over time to cope with the dynamics and heterogeneity of information sources.

Not surprisingly, researchers have been looking for new software designs that cope with such requirements. One promising source of ideas that has been considered in recent years for designing such information integration software is the area of Multi-Agent System (MAS) architectures. They appear to be more flexible, modular and robust than traditional including object-oriented ones. They tend to be open and dynamic in the sense they exist in a changing organizational and operational environment where new components can be added, modified or removed at any time.

To cope with the ever-increasing complexity of the design of software architecture, architectural design has received through the last decade increasing attention as an important field of software engineering.

Practitioners have come to realize that getting an architecture right is a critical success factor for system life-cycle and have recognized the value of making explicit architectural descriptions and choices in the development of new software.

To this end, a number of architectural description languages (ADL) [2] and architectural styles [5] have been proposed for representing and analyzing architectural designs. An *architectural description language* provides a concrete syntax for specifying architectural abstractions in a descriptive notation while an *architectural style* constitutes an intellectually manageable abstraction of system structure that describes how system components interact and work together.

Unfortunately, despite this considerable work, few research efforts have aimed at truly defining styles and description languages for agent architectural design. To fill this gap, we have defined, in the SKwyRL¹ project, architectural styles for multi-agent systems based on an organizational perspective [3] and have proposed in [4] SKwyRL-ADL, an agent architectural description language. This paper continues and integrates this research: it focuses on a multi-agent perspective for designing and specifying information integration architecture based on organizational styles and SKwyRL-ADL. The joint-venture organizational style will be instantiated to design the architecture of the system and the specifications will be expressed in a formal way with SKwyRL-ADL.

The rest of the paper is organized as follows. Section 2 introduces some perspectives of SKwyRL insisting on the BDI model, our ADL and organizational styles. Section 3 describes our multi-agent approach on information integration system development, including

¹ Socio-Intentional ArChitecture for Knowledge Systems and Requirements Elicitation (<http://www.isys.ucl.ac.be/skwyrl/>)

the design of the global architecture with organizational styles, its formal specification with SKwYRL-ADL and the corresponding implementation on an agent-oriented platform. Finally, Section 4 concludes the research.

2 ADL AND STYLES IN SKWYRL

We have detailed in the SKwYRL project an agent ADL called SKwYRL-ADL [4] that proposes a set of abstractions that are fundamental to the description and specification of agent architectures based on the BDI (Belief-Desire-Intention) agent model. To help the reader to understand our ADL specification in the rest of the paper, we briefly present the main elements of SKwYRL-ADL including the BDI agent model. SKwYRL-ADL is composed of two sub-models which operate at two different levels of abstraction: *internal* and *global*. The internal model captures the states of an agent and its potential behavior. The global model describes the interaction among agents that compose the multi-agent architecture. We will also introduce organizational styles through the description of one of them, the joint venture, that will be used later on in the paper.

2.1 The BDI Agent Model

An *agent* defines a system entity, situated in some environment that is capable of flexible autonomous action in order to meet its design objective [9].

An agent can be useful as a stand-alone entity that delegates particular tasks on behalf of a user. However, in the overwhelming majority of cases, agents exist in an environment that contains other agents. Such environment is a agent system that can be defined as an *organization* composed of autonomous and proactive agents that interact with each other to achieve common or private goals [7].

In order to reason about themselves and act in an autonomous way, agents are usually built on rationale models and reasoning strategies that have roots in various disciplines including artificial intelligence, cognitive science, psychology or philosophy. An exhaustive evaluation of these models would be out of the scope of this paper or even this research work. A simple yet powerful and mature model coming from cognitive science and philosophy that has received a great deal of attention, notably in artificial intelligence, is the Belief-Desire-Intention (BDI) model [1]. This approach has been intensively used to study the design of rationale agents

and is proposed as a keystone model in numerous agent-oriented development environments such as JACK [6]. The main concepts of the BDI agent model are in addition to the notion of agent itself we have just explained:

- *Beliefs* that represent the informational state of a BDI agent, that is, what it knows about itself and the world;
- *Desires (or goals)* that are its motivational state, that is, what the agent is trying to achieve;
- *Intentions* that represent the deliberative state of the agent, that is, which plans the agent has chosen for possible execution.

2.2 Internal Model

Figure 1 illustrates the main entities and relationships of the internal model of SKwYRL-ADL. The agent needs knowledge about the environment in order to reach decisions. Knowledge is contained in agents in the form of one of many *knowledge bases*. A Knowledge base consists of a set of *beliefs* that the agent has about the environment and a set of *goals* that it pursues. A belief represents a view of the current environment states of an agent. However, beliefs about the current state of the environment are not always enough to decide what to do. In other words, as well as a current state description, the agent needs some goal information, which describes an environment state that is (not) desirable.

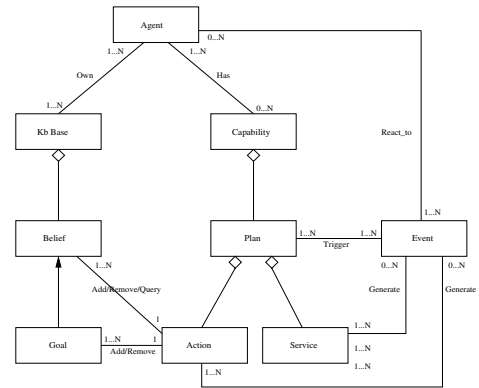


Figure 1: Conceptual Representation of the Internal Model

The intentional behavior of an agent is represented by its *capabilities* to react to *events*. An event is generated either by an *action* that modifies beliefs or adds new goals, or by services provided from another agent. Note that these services are represented in the global model because they involve interaction among agents that compose the agent system.

An event may invoke (trigger) one or more *plans*; the agent commits to execute one of them, that is, it becomes intention. A plan defines the sequence of action to be chosen by the agent to accomplish a task or achieve a goal. An action can query or change the beliefs, generate new events or submit new goals.

2.3 Global Model

Figure 2 conceptualizes the global model which describes the interaction among agents that compose the agent system.

Configurations are the central concept of architectural design, consisting of an interconnected set of *agents*. The topology of a configuration is defined by a set of bindings between provided and required services.

An agent interacts with its environment through an interface composed of sensors and *effectors*. An effector provides to the environment a set of services. Then, a sensor requires a set of services from the environment. A service is an action involving an interaction among agents.

The whole agent system is specified with an *architecture* which contains a set of configurations. An architecture represents the whole system by one or more detailed configuration descriptions.

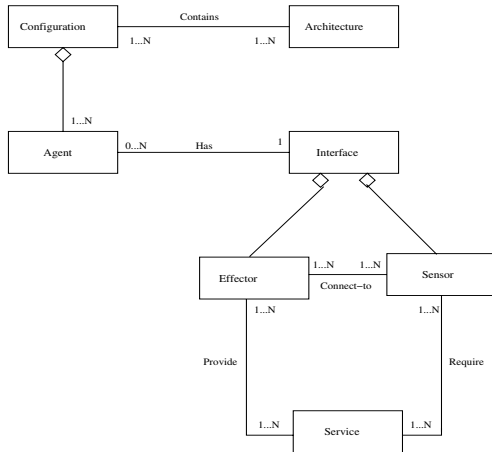


Figure 2: Conceptual Representation of the Global Model

2.4 Multi-Agent Architectural Styles

A key aspect to conduct architectural design in SKwyRL is the specification and use of *organizational styles* (see e.g., [4, 7]) These are socially-based design alternatives inspired by models and concepts from organizational

theories that analyze the structure and design of real-world human organizations. These are the structure-in-5, the joint venture, the chain-of-values, the matrix, the takeover, ...

For instance, the multi-agent architecture we propose in Figure 3 has been designed following and adapting the joint-venture organizational style detailed in [4]. In a few words, the joint-venture organizational style is a meta-structure that defines an organizational system that involves agreement between two or more partners to obtain mutual advantages (greater scale, a partial investment and to lower maintenance costs...). A common actor, the *joint manager*, assumes two roles: a *private interface role* to coordinate partners of the alliance, and a *public interface role* to take strategic decisions, define policy for the private interface, represent the interests of the whole partnership with respect to external stakeholders and ensure communication with the external actors. Each partner can control itself on a local dimension and interact directly with others to exchange resources, data and knowledge.

3 MAS Architecture for Information Integration

GOSIS² is a typical information integration application we have developed using the architectural concepts explained in Section 2. The application provides a Multi-Agent System architecture to support the integration of information coming from different heterogeneous sources.

This section explains how we have used SKwyRL-ADL to formally specify each architectural aspect (belief, goal, plan, action, interface, configuration, service ...) of the application.

3.1 GOSIS Architecture

Figure 3 models the architecture of GOSIS using the *i** model [8] following the joint-venture organizational style we have introduced in Section 2. *i** is a graph, where each node represents an *actor* (or system component) and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an “agreement” (called *dependum*) between two actors: the *dependor* and the *dependee*. The *dependor* is the depending actor, and the *dependee*, the actor who is depended upon. The type of the dependency describes the nature of the agreement. *Goal* dependencies represent delegation of responsibility for fulfilling a goal; *softgoal*

² aGent-Oriented Source Integration System

dependencies are similar to goal dependencies, but their fulfilment cannot be defined precisely; task dependencies are used in situations where the dependee is required.

As show in Figure 3, actors are represented as circles; dependums – goals, softgoals, tasks and resources – are respectively represented as ovals, clouds, hexagons and rectangles; dependencies have the form *depender* → *dependum* → *dependee*.

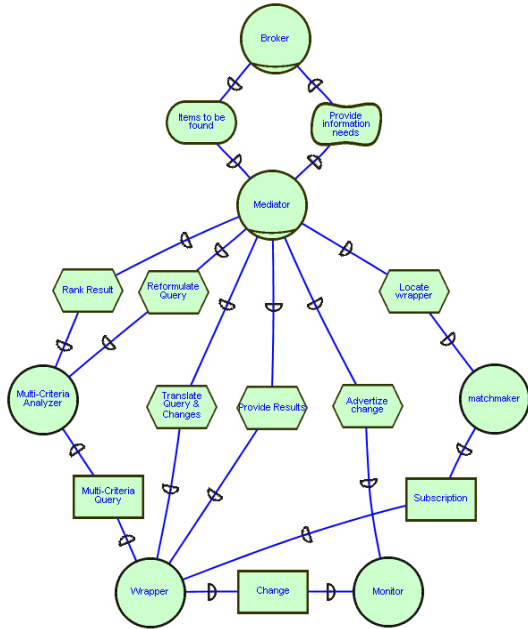


Figure3: The GOSIS Architecture in joint-venture

Figure 3 shows that the mediator plays the role of the joint manager private interface, other joint venture partners are the wrapper, the monitor, the matchmaker and the multi-criteria analyzer. The public interface is assumed by the broker.

When a user wishes to send a request, it contacts the broker agent, which serves as an intermediary to select one or more mediator(s) that can satisfy the user information needs. Then, the selected mediator(s) decomposes the user's query into one or more sub-queries regarding the appropriate information sources, eventually compiles and synthesizes results from the source and returns the final result to the broker.

When the mediator identifies repetitively the same user information needs, this information of interest is extracted from each source, merged with relevant information from the other sources, and stored as knowledge by the mediator. Each stored knowledge constitutes a materialized view the mediator has to maintain up-to-date.

A wrapper and a monitor agents are connected to each information source. The wrapper ensures two roles. It has to translate the sub-query issued by the mediator in the native format of the source and translate the source response in the data model used by the mediator.

The monitor is responsible for detecting changes of interest (e.g., a change which affects a materialized view) in the information source and for reporting them to the mediator. Changes are then translated by the wrapper and sent to the mediator.

It may also be necessary for the mediator to obtain information concerning the localization of a source and its connected wrapper able to provide current or future relevant information. This kind of information is provided by the matchmaker agent, which lets the mediator directly interact with the correspondent wrapper. The matchmaker plays the role of a "yellow-page" agent. Each wrapper advertises its capabilities by subscribing to the yellow page agent. The wrapper that no longer wishes to be advertised can request to be unsubscribed.

Finally, the multi-criteria analyzer reformulates a sub-query (sent by a mediator to a wrapper) through a set of criteria in order to express the user preferences in a more detailed way, and refines the possible domain of results.

3.2 GOSIS Formal Specification

The architecture described in Figure 3 gives an organizational representation of the system-to-be including relevant actors and their respective goals, tasks and resource inter-dependencies. This model can serve as a basis to understand and discuss the assignment of system functionalities but it is not adequate to provide a precise specification of the system details. As introduced in Section 2, SKwyRL-ADL provides a set of formal agent-oriented constructors that allows to detail in a formal and consistent way the software architecture as well as its agent components and their behaviours.

Figure 4 shows a high-level formal description of the *Mediator* agent. Three aspects of this agent component are of concern here: the *interface* representing the interactions in which the agent will participate, the *knowledge base* defining the agent knowledge capacity and the *capabilities* defining agent behaviors.

SkwyRL-ADL allows to work at different levels of architectural abstractions (i.e., different views of the system architecture) to encapsulate different components of the system in independent hierarchical descriptions. For instance, in Figure 4 the *Mediator* agent has a set of

knowledge bases (KB) and a set of capabilities (CP), but the description level chosen here does not specify the details of the beliefs composing the KB or the plans and events composing each capability.

The rest of the section focuses on the *Mediator* agent to give an example of a refinement specification with our ADL for each of the three aspects of the agent: interface, KB and capabilities.

```

Agent:{ Mediator
  Interface:
    Sensor[require(query_translation)]
    Sensor[require(query reformulation)]
    Sensor[require(change_advertizings)]
    ...
    Effector[provide(founded_items)]
  KnowledgeBase:
    Results_KB
    MatchMaker_Info_KB
    DataManagement_KB
    ...
  Capabilities:
    Handle_Request_CP
    Materialized_Views_CP
    Wrapper_Localizaion_CP
  ... }

```

Figure 4: Agent Structure Description of the Mediator

Interface. The agent *interface* consists of a number of effectors and sensors for the agent. Each of them represents an action in which the agent will participate. Each effector provides a service that is available to other agents, and each sensor requires a service provided by another agent. The correspondence between a required and a provided service defines an interaction. For example, the Mediator needs the *query_translation* service that the Wrapper provides.

Such interface definition points two aspects of an agent. Firstly, it indicates the expectations the agent has about the agents with which it interacts. Secondly, it reveals that the interaction relationships are a central issue of the architectural description. Such relationships are not only part of the specification of the agent behavior but reflect the potential patterns of communication that characterize the ways the system reason about itself.

The required query translation service is described in greater detail in figure 5. We can see that the mediator (sender) initiates the service by asking the wrapper (receiver) to translate a query. To this end, the mediator provides to the wrapper a set of parameters allowing to define the contents of this query. Such mediator query is specified as belief with the predicate *search* and the following terms:

```
search(RequestType,ProductType(+),FilteredKeyword(+))
```

Each term represents, respectively, the type of the query (normal advanced in the case of multi-criteria refinement), the type of product and one or many keywords that must be included in or excluded from the results.

```

Service:{Ask(query_translation)
  sender: Mediator
  parameters: rt:RequestType ^ pt:ProductType ^
              fk(+):FilteredKeyword
  receiver: Wrapper
Effect:Add(Translation_Management_KB, search(rt,pt,fk(+))

```

Figure 5: A Service Specification

The service effect indicates that a new search belief is added to the Translation_Management KB of the wrapper.

Knowledge Bases. A *knowledge base* (KB) is specified with a name, a body and a type. The name identifies the KB whenever an agent wants to query or modify them (add or remove a belief). The body represents a set of beliefs in the manner of a relational database schema. It describes the beliefs the agent may have in terms of fields. When the agent acquires a new belief, values for each of its fields are specified and the belief is added to the appropriate KB as a new tuple. The *KB type* describes the kind of formal knowledge used by the agent. A *Closed world* assumes that the agent is operating in a world where every tuple it can express is included in a KB at all times as being true or false. Inversely, in an *open world* KB, any tuple not included as true or false is assumed to be unknown. Figure 6 specifies the Translation_Management_KB:

```

KnowledgeBase: {Translation_Management_KB
KB_body:
  search(RequestType,ProductType,FilteredKeyword(+))
  source_resource(InfoType(+))
  source_modeling(SourceType,Relation(+),Attributes(+))
  dictionary(MediatorTerm,SourceType,Correspondence)
KB_type: closed_world }

```

Figure 6: A Knowledge Base Specification

The '+' symbol means that the attribute is multi-valued.

Capabilities formalize the behavioral elements of an agent. It is composed of plans and events that together define the agent's abilities. It can also be composed of sub-capabilities that can be combined to provide complex behavior.

Figure 7 shows the *Handle_Request* capability of the *Mediator* agent. The body contains the plans the capability handles. The example shows how the request request

Capability: { Handle_Request_CP	post to be
CP_body:	agents. For
Plan DecomNmIRq	composed of
Plan DecomMCRq	se a normal
SendEvent FailUserRq	ulti-criteria
SendEvent FailDecompMCRq	
PostEvent ReadyToHandleRst }	

Figure 7: A Capability Specification

A plan defines the sequence of actions and/or services (i.e., actions that involve interaction with other agents) the agent selects to accomplish a task or achieve a goal. A plan consists of:

- an *invocation condition* detailing the circumstances, in terms of beliefs or goals, that cause the plan to be triggered;
- an *optional context* that defines the preconditions of the plan, i.e., what must be believed by the agent for a plan to be selected for execution;
- the *plan body*, that specifies either the sequence of formulae that the agent needs to perform, a formula being either an action or a service to be executed;
- an *end state* that defines the post-conditions under which the plan succeeds;
- and optionally a set of services or actions that specify what happens when a *plan fails* or *succeeds*.

Figure 8 specifies the *DecompNmIRq* plan that decomposes a normal request.

The supplementary condition about the existence of a *materialized_view* belief is specified by the context. The context is used in the selection of the most appropriate plan in a given situation. When the plan specification does not define a context, the plan is selected to be executed only based on the invocation condition.

As soon as the invocation condition and the context are true, the sequence of actions or services specified in the plan body can be executed. The

DecompNmIRq plan body is composed by an action sequence and a service. The mediator selects from the wrapper beliefs one or many wrappers (wp(+)) able to translate the decomposed sub-queries. A translation service (Ask(query_translation)) is then selected from the selected wrappers.

```

Plan:{ DecomNmIRq
  invoc:
    dd(Request_KB, user_keyword(pt(+),kw(+))
    pt:ProductType From Mediator.Ask(user_info-
    needs).reply_with//
  context:
    ¬ materialized_view(ProductType = pt(+),Keyword = kw(+))
  body:
    ∀ pt : ProductType ∈ user_keyword(pt(+),kw(+)) DO

    action select_wrapper(wrapper(WrapperLocalization,
    TranslationService(+))
  as wp(+): Wrapper
  service:{Ask(query_translation)
    sender: Mediator
    parameters: rt:RequestType ∧ pt:ProductType ∧
    kw(+):Keyword
    receiver: wp(+): Wrapper
  effect: Add(Translation_Management_KB, search(rt,pt,kw(+))
  End-DO
  endstate:
    ∀ pt : ProductType ∈ user_keyword(pt(+),kw(+))
    Add(Translation_Management_KB, search(rt,pt,fk(+))
  succeed:
    count(search(rt,pt,kw(+))
  effect: Add(Request_Kb, old_user_keyword(pt,kw(+)) }

```

Figure 8: A Plan Specification

The plan succeeds when the *endstate* statement is or become true. Moreover, SkwyRL-ADL also specifies what happens when a plan reaches its endstate or fails, further courses of action or service can also be specified to consider what happens next when the plan succeeds or fails. For example, the succeed specification for *DecompNmIRq* counts the number of executions of the current sub-query to identify a potential new materialized view.

Configuration To describe the complete topology of the system architecture, the agents of an architectural description are combined into a SKwyRL *configuration*.

Instances of each agent or service that appear in the configuration must be identified with an explicit and unique name.

The configuration also describes the collaborations (i.e., which agent participates in which interaction) through a one-to-many mapping between provided and required service instances.

Part of the GOSIS configuration with instance declarations and collaborations is given in Figure 9.

“(min)...(max)”. indicates the smallest acceptable integer, and the largest. An omitted cardinality (as is the case

with (*max*) in the broker, mediator and wrapper agents), means no limitation.

```

Configuration GOSIS
  Agent Broker[nb: 1...]
  Agent Mediator[nm: 1...]
  Agent Wrapper[nw: 1...nS] with nS = number of
    information sources
  Agent Monitor[nmo: 1...nS]
  Agent Matchmaker
  Agent Multi-Criteria-analyzer
  Service Tell(query_translation)
  Service Ask(query_translation)
  Service Achieve(result)
  Service Do(result)
  ....
Instances
  BRnb: Broker MEnm: Mediator
  WRnw: Wrapper
  MOnmo: Monitor
  MA: Matchmaker
  MCA: Multi-Criteria-Analyzer
  Tellquerytrans: Tell(query_translation)
  Askquerytrans: Ask(query_translation)
  Achres: Achieve(result)
  Does: Do(result)
  ....
Collaborations
  MEnm.Askquerytrans --- Tellquerytrans.WRnw;
  MEnm.Achres --- Tellres.WRnw;
  MEnm.Asksubs --- Tellsubs.MA;
  ....
End GOSIS

```

Figure 9: The GOSIS Parameterized Configuration

Such a configuration allows for dynamic reconfiguration and architecture resolvability at run-time. Configurations separate the description of composite structures from the description of the elements that form those compositions. This permits reasoning about the composition as a whole and to reconfigure it without having to examine each component of the system

4 CONCLUSION

Nowadays, software engineering for new enterprise application domains such as data integration is forced to build up open systems able to cope with distributed, heterogeneous, and dynamic information issues. Most of these software systems exist in a changing organizational and operational environment where new components can be added, modified or removed at any time. For these reasons and more, multi-agent systems architectures are gaining popularity in that they do allow

dynamic and evolving structures which can change at run-time.

Architectural design has received considerable attention for the past decade which has resulted in a collection of well-understood architectural styles and formal architectural description languages. Unfortunately, these works have focused object-oriented rather than agent-oriented systems. This paper has described an approach based on organizational styles and an agent architectural description language we have defined to design multi-agent systems architectures in the context of information integration system engineering. The paper has proposed a validation of the approach: it has been applied to develop GOSIS, an information integration platform implemented on the JACK agent development environment.

REFERENCES

- [1] M. E. Bratman. Intention, Plans and Practical Reason. Harvard University Press, 1987.
- [2] P. C. Clements. A Survey of Architecture Description Languages. In Proc. of the Eighth International Workshop on Software Specification and Design, Paderborn, Germany, March 1996.
- [3] T. T. Do, S. Faulkner and M. Kolp. Organizational Multi-Agent Architectures for Information Systems. in Proc. of the 5th Int. Conf. on Enterprise Information Systems (ICEIS 2003), Angers, France, April 2003.
- [4] S. Faulkner and M. Kolp. Towards an Agent Architectural Description Language for Information Systems. In Proc. of the 5th Int. Conf. on Enterprise Information Systems (ICEIS 03), Angers, France, April 2003.
- [5] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting Style in Architectural Design Environments. In Proc. of SIGSOFT'94: Foundations of Software Engineering, New Orleans, Louisiana, USA, Dec. 1994.
- [6] JACK Intelligent Agents. <http://www.agent-software.com/>.
- [7] M. Kolp, P. Giorgini, and J. Mylopoulos. An Organizational Perspective on Multi-agent Architectures. In Proc. of the 8th Int. Workshop on Agent Theories, architectures, and languages, ATAL'01, Seattle, USA, Aug. 2001.
- [8] E. Yu. Modeling Strategic Relationships for Process Reengineering, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.
- [9] M. Wooldridge and N.R Jennings, editors. Special Issue on Intelligent Agents and Multi-Agent Systems. Applied Artificial Intelligence Journal. Vol. 9(4), 1996.

Level Construction of Decision Trees in a Partition-based Framework for Classification

Y.Y. Yao, Y. Zhao and J.T. Yao

Department of Computer Science, University of Regina

Regina, Saskatchewan, Canada S4S 0A2

E-mail: {yyao, yanzhao, jtyao}@cs.uregina.ca

Abstract

A partition-based framework is presented for a formal study of consistent classification problems. An information table is used as knowledge representation. Solutions to, and solution space of, classification problems are formulated in terms of partitions. Algorithms for finding solutions are modeled as searching in a space of partitions under a refinement order relation. We focus on a particular type of solutions called conjunctively definable partitions. Two level construction methods for decision trees are investigated. Experimental results are reported to compare the two level construction methods.

1. Introduction

Classification is one of the main tasks in machine learning, data mining and pattern recognition [1, 3, 4]. It deals with classifying labeled objects. Knowledge for classification can be expressed in different forms, such as classification rules, discriminant functions, decision trees and decision graphs.

Classification by decision trees is a popular method. The typical algorithms for decision tree learning are the ID3 algorithm [6] and its descendent, the C4.5 algorithm [7]. Typically, ID3-like algorithms build a decision in a top-down, depth-first mode. Furthermore, the node splitting criteria are based on local optimization. When splitting a node, an attribute is chosen based on only information about this node, but not on any other nodes in the same level. Consequently, different nodes in the same level may use different attributes, and the same attribute may be used at different levels. The use of local optimal criteria makes it difficult to judge the overall quality of the partial decision tree during its construction process.

The main objective of the paper is to study a top-down, breadth-first, level-wise mode for constructing deci-

sion trees. Two types of algorithms are proposed and studied. One is based on local optimization node splitting criteria, and the other is based on global optimization criteria. The former is referred to as the level construction version of ID3 and is denoted by LID3. The latter is in fact a level-wise, reduct based methods and is denoted by k LR. The k LR algorithm combines the methods for constructing decision trees and the methods for searching for reducts [4, 5, 9].

The rest of the paper is organized in two layers. A formal framework for classification is presented in Section 2, which sets the stage for the algorithmic studies. Level construction algorithms are discussed in Section 3 and their experimental evaluations are reported in Section 4. The proposed methods offer a complementary approach to depth-first ID3. The decision trees obtained from the algorithms enables us to see the different aspects of knowledge embedded in data.

2. Consistent Classification Problems

2.1. Information tables

An information table provides a convenient way to describe a finite set of objects by a finite set of attributes. It deals with the issues of knowledge representation for classification problems [5, 11]. An information table S is the tuple:

$$S = (U, At, \mathcal{L}, \{V_a \mid a \in At\}, \{I_a \mid a \in At\}),$$

where U is a finite nonempty set of objects, At is a finite nonempty set of attributes, \mathcal{L} is a language defined by using attributes in At , V_a is a nonempty set of values for a where $a \in At$, and $I_a : U \rightarrow V_a$ is an information function.

Formulas of \mathcal{L} are defined by the following two rules: (i) An atomic formula ϕ of \mathcal{L} is a descriptor $a = v$, where $a \in At$ and $v \in V_a$; (ii) The well-formed formulas (wff) of \mathcal{L} is the smallest set containing the atomic formulas and closed under $\neg, \wedge, \vee, \rightarrow$ and \equiv .

If ϕ is a formula, the set $m_S(\phi)$ defined by

$$m_S(\phi) = \{x \in U \mid x \models \phi\},$$

is called the meaning of the formula ϕ in S . If S is understood, we simply write $m(\phi)$. The meaning of a formula ϕ is the set of all objects having the properties expressed by the formula ϕ . A connection between formulas of \mathcal{L} and subsets of U is thus established.

The notion of definability of subsets in an information table is essential to data analysis. In fact, definable subsets are the basic units that can be described and discussed, upon which other notions can be developed. A subset $X \subseteq U$ is called a definable granule in an information table S if there exists at least one formula ϕ such that $m(\phi) = X$. For a subset of attributes $A \subseteq At$, X is an A -definable granule if there exists at least one formula ϕ_A using only attributes from A such that $m(\phi_A) = X$.

In many classification algorithms, one is only interested in formulas of a certain form. Suppose we restrict the connectives of language \mathcal{L} to only the conjunction connective \wedge . A subset $X \subseteq U$ is a conjunctively definable granule in an information table S if there exists a conjunctive formula ϕ such that $m(\phi) = X$.

2.2. Partitions in an information table

Classification involves the division of the set U of objects into many classes. The notion of partitions provides a formal means to describe classification.

Definition 1 A partition π of a set U is a collection of nonempty and pair-wise disjoint subsets of U whose union is U . The subsets in a partition are called blocks.

Different partitions may be related to each other. A partition π_1 is a refinement of another partition π_2 , or equivalently, π_2 is a coarsening of π_1 , denoted by $\pi_1 \preceq \pi_2$, if every block of π_1 is contained in some block of π_2 . The refinement relation is a partial order, namely, it is reflexive, antisymmetric, and transitive. It defines a partition lattice $\Pi(U)$.

A partition π is called a definable partition in an information table S if every block of π is a definable granule. A partition π is called a conjunctively definable partition if every equivalence class of π is a conjunctively definable granule. Consider two special families of conjunctively definable partitions below.

The first family is the uniformly conjunctively definable partitions, which has been studied extensively in databases [2]. A partition π is called a uniformly conjunctively definable partition in an information table S if, given a subset A of attributes in a certain order, the blocks are further refined in a process that the attributes are one-by-one

added to the conjunctive. $\pi_\emptyset = U$ is the coarsest partition, and π_{At} is the finest partition, for any $A \subseteq At$, we have $\pi_{At} \preceq \pi_A \preceq \pi_\emptyset$.

The second family is the non-uniformly conjunctively definable partitions, which has been used extensively in machine learning [6]. A partition π is called a non-uniformly conjunctively definable partition in an information table S if the partition blocks select their own optimal attributes to further division, according to a consistent selection criteria. This strategy results in that the partition blocks at the same level may use different attributes for refinement partition. Let Π_{tree} be the set of non-uniformly conjunctively definable partitions. The partial order \preceq can be carried over to Π_{tree} . Suppose $\pi_t \in \Pi_{tree}$. It can be easily verified that $\pi_{At} \preceq \pi_t \preceq \pi_\emptyset$.

2.3 Solutions to consistent classification problems

In an information table for classification problems, we have a set of attribute $At = F \cup \{\text{class}\}$. The problem can be formally stated in terms of partitions.

Definition 2 An information table is said to define a consistent classification if objects with the same description have the same class value, namely, for any two objects $x, y \in U$, $I_F(x) = I_F(y)$ implies $I_{\text{class}}(x) = I_{\text{class}}(y)$.

Suppose π is an A -definable partition, that is, each block of π is an A -definable granule. We say that π is a solution to the consistent classification problem, if $\pi \preceq \pi_{\text{class}}$. A solution π is called a most general solution if there does not exist another solution π' such that $\pi \prec \pi' \preceq \pi_{\text{class}}$, where $\pi \prec \pi'$ stands for $\pi \neq \pi'$ and $\pi \preceq \pi'$.

Suppose π is a solution to the classification problem, namely, $\pi \preceq \pi_{\text{class}}$. For a pair of equivalence classes $X \in \pi$ and $C \in \pi_{\text{class}}$ with $X \subseteq C$, we can derive a classification rule $Des(X) \implies Des(C)$, where $Des(X)$ and $Des(C)$ are the formulas that describe sets X and C , respectively.

Let Π_{sol} be the set of all solutions called solution space. The partition π_F is the minimum element of Π_{sol} . For two partitions with $\pi_1 \preceq \pi_2$, if π_2 is a solution, then π_1 is also a solution. For two solutions π_1 and π_2 , $\pi_1 \wedge \pi_2$ is also a solution. The solution space Π_{sol} contains the trivial solution π_F , and is closed under meet \wedge . The solution space is a meet sub-lattice.

In most cases, we are interested in the most general solutions, instead of the trivial solution π_F . In many practical situations, one is satisfied with an approximate solution of the classification problem, instead of an exact solution.

Definition 3 Let $\rho : \pi \times \pi \longrightarrow \mathbb{R}^+$, where \mathbb{R}^+ stands for non-negative reals, be a function such that $\rho(\pi_1, \pi_2)$ measures the degree to which $\pi_1 \preceq \pi_2$ is true. For a threshold

α , a partition π is said to be an approximate solution if $\rho(\pi, \pi_{\text{class}}) \geq \alpha$.

The measure ρ can be defined to capture various aspects of classification. Two such measures are discussed below, they are the *ratio of sure classification* (RSC), and the *accuracy* of classification.

Definition 4 For the partition $\pi = \{X_1, X_2, \dots, X_m\}$, the ratio of sure classification (RSC) by π is given by:

$$\rho_1(\pi, \pi_{\text{class}}) = \frac{\sum_{i=1}^m |\{X_i \in \pi \mid \exists C_j \in \pi_{\text{class}}, X_i \subseteq C_j\}|}{|U|}, \quad (1)$$

where $|\cdot|$ denotes that cardinality of a set. The ratio of sure classification represents the percentage of objects that can be classified by π without any uncertainty. The measure $\rho_1(\pi, \pi_{\text{class}})$ reaches the maximum value 1 if $\pi \preceq \pi_{\text{class}}$, and reaches the minimum value 0 if for all blocks $X_i \in \pi$ and $C_j \in \pi_{\text{class}}$, $X_i \subseteq C_j$ does not hold. For two partitions with $\pi_1 \preceq \pi_2$, we have $\rho_1(\pi_1, \pi_{\text{class}}) \geq \rho_1(\pi_2, \pi_{\text{class}})$.

Definition 5 For the partition $\pi = \{X_1, X_2, \dots, X_m\}$, the accuracy of classification by a partition is defined by:

$$\rho_2(\pi, \pi_{\text{class}}) = \frac{\sum_{i=1}^m |X_i \cap C_{j(X_i)}|}{|U|}, \quad (2)$$

where $C_{j(X_i)} = \arg \max\{|C_j \cap X_i| \mid C_j \in \pi_{\text{class}}\}$. The accuracy of π is in fact the weighted average accuracies of individual rules. The measure $\rho_2(\pi, \pi_{\text{class}})$ reaches the maximum value 1 if $\pi \preceq \pi_{\text{class}}$, and reaches the minimum value $|C_{k_0}|/|U|$, where C_{k_0} is the class with the maximum number of objects. For two partitions with $\pi_1 \preceq \pi_2$, we have $\rho_2(\pi_1, \pi_{\text{class}}) \geq \rho_2(\pi_2, \pi_{\text{class}})$.

Additional measures can also be defined based on the properties of partitions. For example, one may use information-theoretic measures [12].

2.4. Classification as search

Conceptually, finding a solution to a classification problem can be modeled as a search in the space of A -definable partitions under the order relation \preceq . A difficulty with this straightforward search is that the space is too large to be practically applicable. One may avoid such a difficulty in several ways, for instance, only searching the space of conjunctively definable partitions. In particular, in searching the conjunctively definable partitions, two special cases deserve consideration, namely, the space of uniformly conjunctively definable partitions, and the space of non-uniformly conjunctively definable partitions.

Searching solutions in the space of uniformly conjunctively definable partitions can be achieved by rough set based classification methods [5, 9]. Suppose all the attributes of F have same priorities. The goal of solution searching is to find a subset of attributes A so that π_A is a most general solution to the classification problem. The important notions of rough set based approaches are summarized below.

Definition 6 An attribute $a \in A$ is called a core attribute, if $\pi_{F-\{a\}}$ is not a solution, i.e., $\neg(\pi_{F-\{a\}} \preceq \pi_{\text{class}})$.

Definition 7 A subset $A \subseteq F$ is called a reduct, if π_A is a solution and for any subset $B \subseteq A$, π_B is not a solution. That is,

- (i) $\pi_A \preceq \pi_{\text{class}}$;
- (ii) for any proper subset $B \subset A$, $\neg(\pi_B \preceq \pi_{\text{class}})$.

Each reduct provides one solution to the classification problems. There may exist more than one reduct. A core attribute must be presented at each reduct, namely, a core attribute is in every solution to the classification problem. The set of core attributes is the intersection of all reducts. The bias of searching solutions in the space of uniformly conjunctively definable partitions is to find the reduct, a set of individually necessary and jointly sufficient attributes.

The ID3-like algorithms search the space of non-uniformly conjunctively definable partitions. Typically, a classification is constructed in a depth-first manner until the leaf nodes are subsets that consist of elements of the same class with respect to class. By labeling the leaves by the class symbol of class, we obtain a decision tree for classification. The bias of searching solutions in the space of non-uniformly conjunctively definable partitions is to find the shortest tree construction.

One can combine these two searches together, i.e., construct a classification tree by using a reduct set of attributes derived from a reduct-based algorithm.

3. Level Construction of Decision Trees

Two level construction methods are discussed in this section. One is the ID3-like approach, and the other is the reduct-based approach.

3.1. The LID3 algorithm

The ID3 algorithm [6] is perhaps one of the most studied depth-first method for constructing decision trees. It starts with the entire set of objects and recursively divides the set by selecting one attribute at a time, until each node is a subset of objects belong to one class.

A level construction method based ID3 is given below.

LID3: A level construction version of ID3

1. Let $k = 0$.
2. The k -level, $k > 0$, of the classification tree is built based on the $(k - 1)^{th}$ level described as follows:
if a node in $(k - 1)^{th}$ level does not consist of only elements of the same class, **then**
 - 2.1 Choose an attribute based on a certain criterion $\beta : At \rightarrow \mathbb{R}$;
 - 2.2 Divide the node based on the selected attribute and produce the k^{th} level nodes, which are the subsets of that node;
 - 2.3 Label the node by the attribute name, and label the branches coming out from the node by values of the attribute.

The selection criterion used by ID3 is an information-theoretic measures called conditional entropy. Let S denote the set of objects in a particular node at level $(k - 1)$. The conditional entropy of class given an attribute a is denoted by:

$$\begin{aligned}
 H_S(\text{class}|a) &= \sum_{v \in V_a} P_S(v) H(\text{class}|v) \\
 &= - \sum_{v \in V_a} P_S(v) \sum_{d \in V_{\text{class}}} P_S(d|v) \log P_S(d|v) \\
 &= - \sum_{d \in V_{\text{class}}} \sum_{v \in V_a} P_S(d, v) \log P_S(d|v), \quad (3)
 \end{aligned}$$

where the subscript S indicates that all quantities are defined with respect to the set S . An attribute with the minimum entropy value is chosen to split a node.

For the information Table 1, we obtain a decision tree shown in Figure 1, and the analysis of RSC and accuracy is summarized in Table 2.

	A	B	C	D	class
1	a_1	b_1	c_1	d_2	-
2	a_1	b_1	c_2	d_2	-
3	a_1	b_2	c_1	d_1	+
4	a_1	b_2	c_2	d_1	+
5	a_2	b_1	c_1	d_2	-
6	a_2	b_1	c_2	d_1	-
7	a_2	b_2	c_1	d_2	-
8	a_2	b_2	c_2	d_1	+
9	a_3	b_1	c_1	d_2	+
10	a_3	b_1	c_2	d_1	-
11	a_3	b_2	c_1	d_1	+
12	a_3	b_2	c_2	d_1	+

Table 1. An information table

	Rules	RSC	Accuracy
$k = 1$ B	$b_1 \Rightarrow 5/6 -$ $b_2 \Rightarrow 5/6 +$	0.00	0.83
$k = 2$ ABD	$b_1 \wedge a_1 \Rightarrow 2/2 -$ $b_1 \wedge a_2 \Rightarrow 2/2 +$ $b_1 \wedge a_3 \Rightarrow 1/2 +$ $b_2 \wedge d_1 \Rightarrow 5/5 +$ $b_2 \wedge d_2 \Rightarrow 1/1 -$	0.83	0.92
$k = 3$ $ABCD$	$b_1 \wedge a_3 \wedge c_1 \Rightarrow 1/1 +$ $b_1 \wedge a_3 \wedge c_2 \Rightarrow 1/1 -$	1.00	1.00

Table 2. Rules generated by ID3

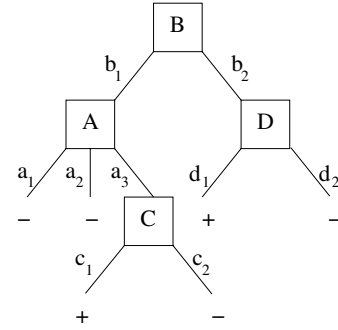


Figure 1. An ID3 decision tree

3.2. The k LR algorithm

Recall that a reduct is a set of individually necessary and jointly sufficient attributes that correctly classify the objects. An algorithm for finding a reduct can be easily extended into a level construction method for decision trees called k LR.

k LR: A reduct-based level construction method

1. Let $k = 0$.
2. The k -level, $k > 0$, of the classification tree is built based on the $(k - 1)^{th}$ level described as follows:
if there is a node in $(k - 1)^{th}$ level that does not consist of only elements of the same class **then**
 - 2.1 Choose an attribute based on a certain criterion $\gamma : At \rightarrow \mathbb{R}$;
 - 2.2 Divide all the inconsistent nodes based on the selected attribute and produce the k^{th} level nodes, which are subsets of the inconsistent nodes;
 - 2.3 Label the inconsistent nodes by the attribute name, and label the branches coming out from the inconsistent nodes by the values of the attribute.

Note that when choosing an attribute, one needs to consider all the inconsistent nodes. In contrast, LID3 only con-

	Rules	RSC	Accuracy
$k = 1$ B	$b_1 \Rightarrow 5/6 -$ $b_2 \Rightarrow 5/6 +$	0.00	0.83
$k = 2$ BD	$b_1 \wedge d_1 \Rightarrow 2/2 -$ $b_1 \wedge d_1 \Rightarrow 3/4 -$ $b_2 \wedge d_1 \Rightarrow 5/5 +$ $b_2 \wedge d_2 \Rightarrow 1/1 -$	0.67	0.92
$k = 3$ ABD	$b_1 \wedge d_2 \wedge a_1 \Rightarrow 2/2 -$ $b_1 \wedge d_2 \wedge a_2 \Rightarrow 1/1 -$ $b_1 \wedge d_2 \wedge a_3 \Rightarrow 1/1 +$	1.00	1.00

Table 3. Rules generated by k LR

siders one inconsistent node at a time.

Conditional entropy can also be used as the selection criterion γ . In this case, the subset of examples considered at each level is the union of all inconsistent nodes. Let $A_{(k-1)}$ be the set of attributes used from level 0 to level $(k-1)$. The next attribute a for level k can be selected based on the following conditional entropy:

$$H(\text{class} | A_{(k-1)} \cup \{a\}). \quad (4)$$

The use of $A_{(k-1)}$ ensures that all inconsistent nodes at level $k-1$ are considered in the selection of level k attribute [9].

The decision trees generated by the k LR algorithm are level-constructed trees. The idea is similar to oblivious decision trees, in which all nodes at the same level test the same attributes according to a given order. While the order is not given, we can use the function $\gamma : At \rightarrow \mathbb{R}$ to decide an order. The criterion γ can be one of the information measures, for example, conditional entropy (as shown above) or mutual information, which indicate how much information the attributes contribute to the decision attribute **class**; or the statistical measures, for example, the χ^2 test or binomial distribution, which indicates the dependency level between the test attribute and the decision attribute **class**. We can get such an order by testing all the attributes. However, we need to update the order level by level. There are two reasons for level-wise updating. First, in respect that some nodes of a classification tree are intended to halt the partition when the solutions or the approximate solutions are found. The search space is possibly changed for different levels. Second, for each test that partitions the search space into uneven-sized blocks, the value of function γ is the sum of the function value of γ for each block multiplies the probability distribution of the block.

Consider the earlier example, based on the conditional entropy, the decision tree built by k LR algorithm is shown in Figure 2. The analysis of RSC and accuracy is given by Table 3.

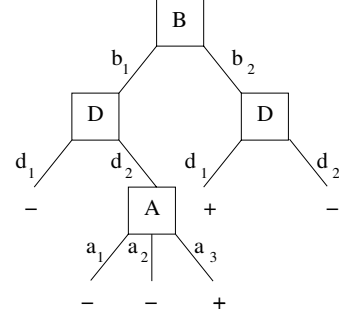


Figure 2. A k LR decision tree

Comparing these two decision trees in Figures 1 and 2, we notice that k LR decision tree can construct a tree possessing the same RSC and accuracy as the LID3 decision tree with fewer attributes involved. In this example, the set of attributes $\{A, B, D\}$ is a reduct, since $\pi_{\{A, B, D\}} \preceq \pi_{\text{class}}$, and for any proper subset $X \subset \{A, B, D\}$, $\neg(\pi_X \preceq \pi_{\text{class}})$.

4. Experimental evaluations

In order to evaluate level construction methods, we choose some well-known datasets from UCI machine learning repository [10]. SGI's MLC++ utilities 2.0 [8] is used to discretize the datasets into categorical attribute sets. Set the accuracy threshold $\alpha = 100\%$.

Dataset 1: Credit - 690 training objects, 14 attributes and 2 classes. Using all the attributes, neither LID3 nor k LR can 100% consistently classify all the training instances. k LR discovers that one attribute contributes little to the classification, namely, we can have the same RSC and accuracy values with only 13 attributes. k LR generates more rules than its counterpart. However comparing the trees where the same number of attributes are used, k LR obtains higher RSC and accuracy.

Dataset 2: Vote - 435 training objects, 16 attributes and 2 classes. It achieves 100% of RSC and accuracy for classification. In the case of LID3, the total tree length is 8 levels, but 15 attributes were required for 100% of RSC and accuracy. In the case of k LR, we can reach the same level of RSC and accuracy by a 9-level-tree with only 9 related attributes.

Dataset 3: Cleve - 303 training objects, 13 attributes and 2 classes. It cannot be 100% consistently classified either by all its 13 attributes. In this dataset, the k LR tree is short than the LID3 tree. The k LR tree discovered consists of 11 attributes. This number is less than that is used for constructing a full LID3 tree. This observation is shown in all the other experiments.

The experimental results of the above three datasets are

Dataset 1		
Goal	LID3	kLR
$Accu. \geq 85.00\%$	$k = 1$ (1 attr)	$k = 1$ (1 attr)
$Accu. \geq 90.00\%$	$k = 4$ (14 attrs)	$k = 5$ (5 attrs)
$Accu. \geq 95.00\%$	$k = 6$ (14 attrs)	$k = 8$ (8 attrs)
$Accu. = 98.55\%$	$k = 11$ (14 attrs)	$k = 13$ (13 attrs)
$RSC \geq 85.00\%$	$k = 6$ (14 attrs)	$k = 8$ (8 attrs)
$RSC \geq 90.00\%$	$k = 7$ (14 attrs)	$k = 9$ (9 attrs)
$RSC \geq 95.00\%$	$k = 10$ (14 attrs)	$k = 12$ (12 attrs)
$RSC = 95.80\%$	$k = 11$ (14 attrs)	$k = 13$ (13 attrs)
Dataset 2		
Goal	LID3	kLR
$Accu. \geq 95.00\%$	$k = 1$ (1 attr)	$k = 1$ (1 attr)
$Accu. = 100.00\%$	$k = 8$ (15 attrs)	$k = 9$ (9 attrs)
$RSC \geq 95.00\%$	$k = 5$ (11 attrs)	$k = 6$ (6 attrs)
$RSC = 100.00\%$	$k = 8$ (15 attrs)	$k = 9$ (9 attrs)
Dataset 3		
Goal	LID3	kLR
$Accu. \geq 85.00\%$	$k = 3$ (6 attrs)	$k = 3$ (3 attrs)
$Accu. \geq 90.00\%$	$k = 5$ (11 attrs)	$k = 6$ (6 attrs)
$Accu. \geq 95.00\%$	$k = 6$ (12 attrs)	$k = 8$ (8 attrs)
$Accu. = 98.35\%$	$k = 13$ (12 attrs)	$k = 11$ (11 attrs)
$RSC \geq 80.00\%$	$k = 6$ (12 attrs)	$k = 7$ (7 attrs)
$RSC \geq 85.00\%$	$k = 7$ (12 attrs)	$k = 8$ (8 attrs)
$RSC \geq 90.00\%$	$k = 8$ (12 attrs)	$k = 9$ (9 attrs)
$RSC = 94.72\%$	$k = 13$ (12 attrs)	$k = 11$ (11 attrs)

Table 4. Experimental results of the datasets used in this paper

reported in Table 4.

From the results of experiments, we can have the following observations. The difference of local and global selection causes different tree structures. Normally, LID3 may obtain a shorter tree. On the other hand, if we restrict the height of decision trees, LID3 may use more attributes than kLR. With respect to the RSC measure, LID3 tree is normally better than kLR tree at the same level. With respect to the accuracy measure, LID3 tree is not substantially better than kLR tree at the same level. With respect to different levels of two trees with the same number of attributes, kLR obtains much better accuracy and RSC than LID3. The main advantage of kLR method is that it uses fewer number of attributes to achieve the same level of accuracy.

5. Conclusion

The contribution of this paper is twofold, the development of a formal model and algorithms for level construction methods for building decision trees. The formal framework is based on partitions in an information table. Within the framework, we are able to define precisely and concisely many fundamental notions. The concepts of solu-

tion and solution space are discussed. The structures of several search spaces are studied. Two level construction methods are suggested: a breath-first version of ID3 called LID3, which searches the space of non-uniformly conjunctively definable partitions; and a reduct-based method called kLR, which searches solutions in the space of uniformly conjunctively definable partitions. Experimental results are reported to compare these two methods. They show that one needs to pay more attention to the less studied level construction methods.

References

- [1] Duda, R.O. and Hart, P.E. *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
- [2] Lee, T.T. An information-theoretic analysis of relational databases - part I: data dependencies and information metric, *IEEE Transactions on Software Engineering*, **SE-13**, 1049-1061, 1987.
- [3] Michalski, J.S., Carbonell, J.G., and Mitchell, T.M. (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Morgan Kaufmann, Palo Alto, CA, 463-482, 1983.
- [4] Mitchell, T.M., *Machine Learning*, McGraw-Hill, 1997.
- [5] Pawlak, Z., *Rough Sets: Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht, 1991.
- [6] Quinlan, J.R., Induction of decision trees, *Machine Learning*, **1**, 81-106, 1986.
- [7] Quinlan, J.R., *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, Inc., 1993.
- [8] SGI's MLC++ utilities 2.0: the discretize utility. <http://www.sgi.com/tech/mlc>
- [9] Wang, G.Y., Yu, H. and Yang, D.C., Decision table reduction based on conditional information entropy, *Chinese Journal of Computers*, **25(7)**, 2002. 3.
- [10] UCI Machine Learning Repository. <http://www1.ics.uci.edu/~mllearn/MLRepository.html>
- [11] Yao, J.T. and Yao, Y.Y., Induction of classification rules by granular computing, *Proceedings of International Conference on Rough Sets and Current Trends in Computing*, 331-338, 2002.
- [12] Yao, Y.Y., Information-theoretic measures for knowledge discovery and data mining, *Entropy Measures, Maximum Entropy and Emerging Applications*, Karmeshu (Ed.), Springer, Berlin, 115-136, 2003.

Mapping CM^3 : Upfront Maintenance on CGE&Y's Process Model

Mira Kajko-Mattsson,
SML: Upfront Maintenance
Dept of Computer and Systems Sciences
Stockholm Univ. & Royal Inst. of
Technology, Sweden
mira@dsv.su.se

Karin Ericsson, Zsafia Szalkai
Dept of Computer and Systems Sciences
Stockholm Univ. & Royal Inst. of
Technology, Sweden
[\[zsafia-s; karin-er\]@fc.dsv.su.se](mailto:[zsafia-s; karin-er]@fc.dsv.su.se)

ABSTRACT

We have created a process model for managing corrective maintenance requests at the front-end support level. Our model is called CM^3 : Upfront Maintenance. It was developed at two ABB organisations. In this paper, we check its applicability in the context of another organisation – Cap Gemini Ernst & Young.

1. Introduction

Evolution and maintenance may be conducted by one or several organisations, departments, teams and/or even individuals. Usually, it is performed by several co-operating organisations, building a so-called virtual IT enterprise. As depicted in Figure 1, such an enterprise is logically divided into three organisational levels, where each level plays a clearly defined role within evolution and maintenance. The levels are: (1) customer, (2) front-end support, and (3) back-end support. The customer uses software products and states new requirements for evolving and maintaining them. The front-end support assists the customer in a daily operation of the software products. The back-end support evolves and maintains the products according to the requirements as requested by the customer. Many times, it is the organisations at the back-end process level that have developed the system.

Front-end support organisations are the face of the back-end development and maintenance organisations towards their customers. They assist their customers in the operation of the software or integrated software and hardware products. They also assist the back-end evolution and maintenance organisations in communicating maintenance demands requested by the customers.

Front-end support processes are one of the most diverse and complex to define. So far, they have not achieved enough attention. Most research effort has been put into the development and maintenance processes and the improvement of these processes. Very little research however has been done within their supporting front-end

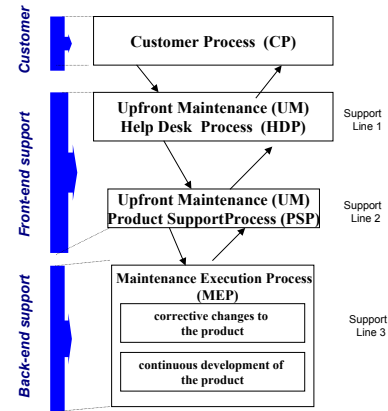


Figure 1. Support levels [7]

support processes – the processes that extensively mediate or eliminate the need for changes in software systems.

Today, we have very little knowledge of and insight into the front-end support processes, despite the fact that they are the dominating cost within evolution and maintenance [2]. To remedy this, we have created a process model of front-end support, called CM^3 : Upfront Maintenance. This model is part of a greater model called *Corrective Maintenance Maturity Model*, abbreviated as CM^3 . In this paper, we present one of its parts dealing with its process phases and activities. This part was developed at two ABB organisations [6]. We map its phases and activities on another real-life industrial front-end support process utilised by one department at Cap Gemini Ernst & Young (CGE&Y). Our goal is to find out whether our model is applicable in the context of other organisations.

The remainder of this paper is the following. In Section 2, we describe our contribution. Section 3 describes CM^3 : Upfront Maintenance process phases and its activities. In Section 4, we map them on CGE&Y's process phases and activities. Finally, in Sections 5 and 6, we make final remarks and suggestions for future work.

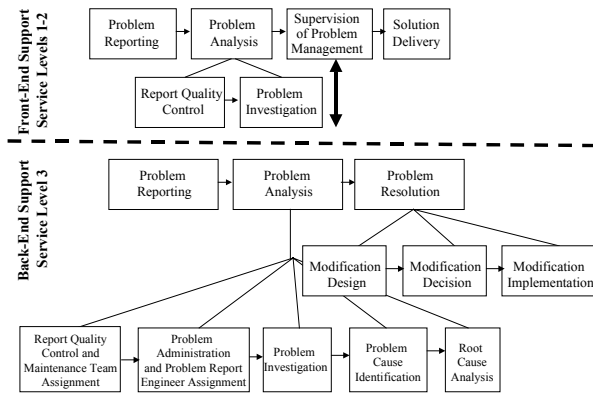


Figure 2. CM³: process phases

2. Contribution

Right now, we are in the process of defining a process model called *Corrective Maintenance Maturity Model: Upfront Maintenance*, abbreviated as CM³:*Upfront Maintenance*. This model is limited to problem management within corrective maintenance at the front-end support level. By upfront maintenance, we mean part of the front-end support conducting maintenance-related tasks. The model was developed by the lead author of this paper, when studying the front-end and back-end processes at two major organisations supporting real-time applications: *ABB Robotics*, *ABB Service* in the years 2000-2001 [6].

In this paper we check the applicability of CM³:*Upfront Maintenance* at CGE&Y. For this purpose, we studied CGE&Y's front-end problem management process called *Problem Handling* and mapped its activities on CM³:*Upfront Maintenance* activities. The CGE&Y process was developed to manage customer requests for only one particular customer and product (administrative product). It is executed by only one department. Hence, it is not representative of the standard process model that has been globally defined for the whole CGE&Y.

3. CM³: Upfront Maintenance

The upper part of Figure 2 depicts CM³:*Upfront Maintenance* process phases. The bottom part shows CM³:*Problem Management* at the back-end support process level [5]. Regarding the upfront maintenance process model, it consists of four main phases. They are (1) *Problem Reporting* phase, (2) *Problem Analysis* phase, (3) *Problem Management Supervision* phase, and (4) *Solution Delivery* phase. These phases and their activities are described below. They are also listed in Table 1.

Problem Reporting Phase: During this phase, problems get reported to the support organisation by the

problem submitter. Usually, the problem submitter is the customer filling in the problem report with relevant data describing the problem. It may however be a support engineer who does it for some customer's account. The support engineer may also report an internally encountered problem. The list of activities for this process phase is defined in Table 1. If the problem is reported internally, then the support engineer must conduct additional activities required for future analysis and control of the support process (see CM³ Activities A-1.20-21). Hence, not all activities in this phase apply to external problem submitters. At this phase, the problem report acquires the status value "*Reported*".

Problem Analysis Phase: During this phase, the front-end support engineers attempt to recreate the reported problems in order to confirm that the reported problem is really a problem. We divide this phase into the two sub-phases: (1) *Report Quality Control and Problem Owner Assignment* phase, and (2) *Problem Investigation* phase.

During the first sub-phase - *Report Quality Control and Problem Owner Assignment*, one controls the quality of the reported data. A problem report that is incorrect, incomplete and inconsistent may lead to inappropriate understanding of the software problem, thus obstructing the problem resolution process and decreasing support productivity [1, 3]. Therefore, after a problem has been reported, the first task to be performed by the upfront maintenance process is to check the correctness of the reported data. If it is not satisfactory, the problem submitter should be contacted for clarification and supplementary additions [1, 5]. Finally, one assigns the problem report to some support engineer. This assignment is dependent on the expertise required for attending to the problem report. The expertise may be found within the same premises of the company, or it may be found within another support company on the same support line level, sometimes even situated within a different time zone. At this sub-phase, the problem report acquires the status value "*Assigned to Problem Owner*".

During *Problem Investigation* sub-phase, the problem report owner attempts to confirm the presence of a software problem. The key focus is to reproduce the problem mainly by executing software. The problem owner must identify whether the reported problem is unique or whether it is a duplicate. Hence, the exact steps taken during this stage may vary depending on the nature of the problem. If the problem is identified as a new problem, then the support engineer is obliged to report it to the next support level (the next front-end or back-end support process level – see Figure 1). If the problem shows to be an already known problem (a duplicate), then the engineer may close its resolution, and link the duplicate problem report to the master report (the report communicating the unique problem). In some cases, however, she may complement the original master report

Table 1. The CM³: Upfront Maintenance process phases and activities

<u>Problem Reporting</u>			
A-1.1:	Describe the problem	A-1.10:	Classify the problem report as internal or external.
A-1.1.1:	Give a general textual description of the problem.	A-1.11:	Assign the submitter's judgement of the problem severity and priority.
A-1.1.2:	Describe the problem effect(s) and consequence(s).	A-1.12:	Identify the activity during which the software problem was encountered.
A-1.1.3:	Describe the symptoms of the problem.	A-1.13:	Record the date and time when the problem was encountered, if relevant.
A-1.1.4:	Describe the problem conditions.	A-1.14:	Identify problems related to the reported problem, if any.
A-1.1.5:	Indicate the reproducibility of the problem.	A-1.15:	Describe (a) work-around(s), if any.
A-1.1.5.1:	Classify the problem as reproducible or non-reproducible.	A-1.16:	Suggest solutions to the reported problem, if any.
A-1.1.5.2:	Indicate the repeatability of the problem.	A-1.17:	Enter problem report data into the organisation-wide problem report database.
A-1.1.5.3:	Describe how to reproduce the problem.	A-1.18:	Assign a unique identifier to the problem report.
A-1.1.5.4:	Describe the alternative execution path(s) to the problem.	A-1.19:	Record the date and time when the problem entered the problem report repository and tracking system.
A-1.1.6:	Attach the relevant file(s).	A-1.20:	Record the additional activities/tasks performed during the activity.
A-1.2:	Write a short summary (title) of the problem For mnemonic identification and browsing purposes.	A-1.21:	Record the effort and resources of the activity "Problem Reporting".
A-1.3:	Identify the support category.	A-1.22:	Assign the status "Reported" to the problem report.
A-1.4:	Identify type of a problem.	A-1.23:	Record the date and time when the problem report was assigned the status "status name".
A-1.5:	Identify the product in which the problem was encountered.		
A-1.6:	Identify the product release ID in which the problem was encountered.		
A-1.7:	Identify the product component/function in which the problem was encountered.		
A-1.8:	Identify the environment of the product.		
A-1.9:	Identify the problem submitter.		
<u>Report Quality Control and Problem Owner Assignment</u>			
A-2.1.1:	Study the problem report.	A-2.1.8:	Record the additional activities/tasks performed during the activity "Report Quality Control and Problem Owner Assignment".
A-2.1.2:	Check the correctness, consistency and completeness of the reported data.	A-2.1.9:	Record the effort and resources of the activity "Report Quality Control and Problem Owner Assignment".
A-2.1.3:	Confirm the problem. Comment: At this step, you just check whether the report communicates a problem.	A-2.1.10:	Assign the status "Assigned to Maintenance Team" to the problem report.
A-2.1.4:	Check whether the problem is unique.	A-2.1.11:	Record the date and time when the problem report was assigned the status "Assigned to Problem Owner".
A-2.1.5:	Revise the maintenance category.		
A-2.1.6:	Assign the maintainer's judgement of problem severity and priority.		
A-2.1.7:	Assign the problem report to the relevant maintenance team.		
<u>Problem Investigation</u>			
A-2.2.1:	Study the problem report		problem in the master- and the duplicate problem report, if necessary.
A-2.2.2:	Check the correctness, consistency and completeness of the reported data.	A-2.2.3.2.5:	Check the progress of the problem resolution.
A-2.2.3:	Confirm the problem.	A-2.2.3.2.6:	Link the duplicate problem report to its unique correspondence (master report).
A-2.2.3.1:	If the problem report owner suspects that the problem is unique, he conducts the steps relevant for confirming that the problem is new to the support organisation.	A-2.2.3.2.7:	Close the management of the duplicate software problem report.
A-2.2.3.1.1:	Read the problem description.	A-2.2.3.4:	Report to the customer on the status of the problem investigation.
A-2.2.3.1.2:	Study the software system.	A-2.2.3.5:	Record the results of problem investigation.
A-2.2.3.1.3:	Designate (Revise the designation of) an appropriate version(s) of the product in which the problem will be investigated.	A-2.2.3.6:	Suggest solutions to the reported problem, if any.
A-2.2.3.1.4:	Create a pertinent execution environment for the version(s) of the product in which the problem will be investigated.	A-2.2.3.7:	Investigate whether there is a work-around.
A-2.2.3.1.5:	Define a set of test cases required for problem investigation.	A-2.2.3.8:	Identify the support category.
A-2.2.3.1.6:	Execute the software system until the problem is recreated.	A-2.2.3.9:	Revise the maintainer's judgement of the problem priority and severity.
A-2.2.3.1.7:	Revise and complement the description of the software problem, if necessary.	A-2.2.3.10:	Submit additional problems encountered during the activity "Problem Investigation".
A-2.2.3.1.8:	Check and record whether the problem is unique or a duplicate.	A-2.2.3.11:	Record the additional activities/tasks performed during the activity "Problem Investigation".
A-2.2.3.1.8.1:	Use a list of known problems in order to determine the uniqueness of the software.	A-2.2.3.12:	Record the effort and resources of the activity "Problem Investigation".
A-2.2.3.1.8.2:	Identify and classify the symptoms of the reported problem.	A-2.2.3.13:	Assign the status "Problem Investigated" to the problem report that has been identified as a unique report, and the status "Cancelled" to the duplicate problem report.
A-2.2.3.2:	If the report owner suspects out of the problem description that the problem report is a duplicate, then he conducts the steps relevant for managing duplicate problem reports	A-2.2.3.14:	Record the date and time when the problem report was assigned the status "Problem Investigated".
A-2.2.3.2.1:	Read the problem description.	A-2.2.3.15:	Deliver the results to the back-end process level (maintenance execution process level). This step is relevant in cases when the problem is unique or when a master report needs to be updated with additional data from the duplicate problem report.
A-2.2.3.2.2:	Identify the master report.	A-2.2.3.16:	Report to the customer on the status of the problem resolution.
A-2.2.3.2.3:	Check and record whether the problem is unique or duplicate. If necessary, repeat the steps A-2.2.3.1.2 - A-2.2.3.1.6.		
A-2.2.3.2.3.1:	Use a list of known problems in order to determine the uniqueness of the software.		
A-2.2.3.2.3.2:	Identify and classify the symptoms of the reported problem.		
A-2.2.3.2.4:	Revise and complement the description of the software		
<u>Problem Supervision</u>			
A-3.1:	Continuously supervise the management of the problem at the back-end process level via the process supporting tool. Comment: The engineer is either automatically notified about the progress by the supporting tool or she does it actively on her own.	A-3.2.4:	engineers study the modification designs by listening to the oral presentations of the back-end support engineers.
A-3.2:	Attend CCB meetings	A-3.2.5:	Analyse and comment on the modification design from the customer's perspective.
A-3.2.1:	Study the problem description. Comment: The more the back-end support engineer works on a problem, the more understanding she gains. At the back-end process level, the problem descriptions are the closest possible interpretations of the true nature and complexity of a software problem. They may differ from the original problem descriptions as reported to the front-end support. Hence, it is important that the front-end support engineer reads the problem description anew. All the CCB-members should do so as well.	A-3.3:	Analyse and comment the modification design from the support perspective.
A-3.2.2:	Understand the underlying cause (defect).	A-3.3.1:	Conduct system testing
A-3.2.3:	Study the modification design (problem solution). Comment: Usually, during the CCB meetings, the support	A-3.4:	Conduct steps relevant for the system tests, defined in CM ³ Testing.
		A-3.5:	Comment: At this phase, the support engineer checks whether the reported software problem has been resolved
		A-3.6:	Record the additional activities/tasks performed during the activity "Problem Management Supervision".
		A-3.7:	Record the effort and resources of the activity "Problem Management Supervision".
			Assign the status "Under Resolution" to the problem report.
			Record the date and time when the problem report was assigned the status "status name".
<u>Solution Delivery</u>			
A-4.1:	Deliver the release containing the problem solution to the customers affected by the problem.	A-4.2:	Announce to other customers that the release containing the problem solution is ready for delivery.

with the extra information provided in duplicate problem report, if she believes that this extra information would aid in better problem understanding and thereby in more efficient problem investigation. At this sub-phase, the problem report acquires the status value “*Problem Investigated*”.

Problem Management Supervision Phase: During the *Problem Management Supervision* phase, the problem report is being managed by the back-end support engineers. This fact however should not release the front-end support engineers from abandoning the problem report ownership. The ownership should continue till the problem has been solved, that is, till the tested problem solution has been delivered to the customer.

As depicted in Figure 2, during this phase, the front-end support engineer should continuously supervise the management of the problem at the back-end process level. She is either automatically notified about its progress via the tool or she does it on her own by regularly checking the contents of the support database. She should also attend the *Change Control Board (CCB)* meetings during which the problems and problem solutions (modifications designs) are being discussed. Finally, the front-end support engineer should assist in system testing of the releases containing solutions to the reported software problems. At this phase, the problem report acquires the status value “*Under Resolution*”.

Solution Delivery Phase: During the *Solution Delivery* phase, the release containing one or several solutions is being delivered to the customers affected by the resolved problems. It is also announced as available to all the other customers.

4. Mapping CM³: Upfront Maintenance on the CGE&Y’s process

In this section, we compare the CM³: *Upfront Maintenance* and CGE&Y process activities. For this purpose, we have identified a number of mapping criteria. They are specified and motivated in Sections 4.1-4.3 together with our mapping results. Finally, we would like to point out that due to the space scarcity, we have chosen to map only a subset of activities, the activities that we feel are the most relevant and important for defining and managing the front-end support operation.

4.1 Problem Reporting Phase

The mapping criteria and results for the “*Problem Reporting Phase*” are the following:

Mapping Criteria 1: Support demands are recorded and categorised (CM³: Process Activities: A-1.3, A-1.17 in Table 1): Support organisations can acquire information about the maintenance demands and needed changes only if they are properly recorded. Hence, all customer

demands, including software problems, should be recorded and categorised. Achieving control over the number of customer demands, and their distribution by types greatly aids in monitoring the support process, assessing its maintenance scope and in planning for future work.

CGE&Y records all the support demands reported to them in a supporting database based tool called Artologik (see CGE&Y Activity A-1.8 in Table 2)¹. Hence, they have control over each single customer demand, which in turn provides a basis for defining the scope of support and a basis for monitoring the support process.

CGE&Y does also distinguish between problem reports and other types of support demands. They do it via the priority value (see Activity 1.6 in Table 2). The demands concerning the corrective changes are assigned priority values 1-3, whereas the demands concerning enhancements are assigned priority value 4.

Mapping Criteria 2: Problems are exhaustively described (CM³: Process Activities: A-1.1 (A1.1.1-A1.1.6 and A-1.12 in Table 1): Problem reports are requirement specifications within corrective maintenance. They communicate requirements for change. A proper problem description is the most important prerequisite for effective problem resolution. A poor, sketchy, or misleading description may lead to an enormous effort to recreate the problem, and thereby substantially retard the problem resolution process. For this reason, a problem report should be a clear, complete and correct description of a software problem. The problem has to be communicated in a structured and disciplined way.

To aid in minimising the reporting time for problem submitters and in maximising the quality of the reported data, the maintenance organisation should give guidance to their problem submitters on how to provide and structure problem description data. This may be done by creating a template of problem descriptions. Such a template should clearly designate the fields relevant for providing general descriptions of a problem, problem effects and consequences, symptoms, problem conditions, specifications of how to reproduce the problem, and opportunities for attaching relevant files.

CGE&Y today collects all the information relevant for describing software problems (see CGE&Y activity A-1.4 in Table 2). However, they have not elaborated any detailed template for problem descriptions. All the relevant information is to be provided in one and only one textual field. This field is not structured in any particular way.

Mapping Criteria 3: Temporal Aspects of a Problem are recorded (CM³: Process Activities: A-1.13 and A-1.19 in Table 1): For some problems, it may be important to identify the process or operation step during which the

¹ This tool is owned and managed by their customer organisation. It does not belong to the ordinary CGE&Y’s standard tool portfolios utilised for managing customer demands.

Table 2. Front-end support process activities at the CGE&Y process studied

<u>Problem Reporting</u>	
A-1.1:	Identify the problem submitter.
A-1.2:	Designate the back-end support level to which the problem report is sent for attendance.
A-1.3:	Write the title of the problem.
A-1.4:	Describe the problem.
A-1.4.1:	Give a general textual description of the problem.
A-1.4.2:	Describe the problem effect(s) and consequence(s).
A-1.4.3:	Describe the symptoms of the problem.
A-1.4.4:	Describe the problem conditions.
A-1.4.5:	Indicate the reproducibility of the problem.
A-1.4.5.1:	Classify the problem as reproducible or non-reproducible.
A-1.4.5.2:	Indicate the repeatability of the problem.
A-1.4.5.3:	Describe how to reproduce the problem.
A-1.4.5.4:	Describe the alternative execution path(s) to the problem.
A-1.4.6:	Attach the relevant file(s).
A-1.5:	Identify the product in which the problem was encountered.
A-1.6:	Assign the submitter's judgement of the problem priority.
A-1.7:	Identify the additional contact persons that may be contacted during the problem handling process.
A-1.8:	Enter the problem report data into the 'CGE&Y' problem report database.
A-1.9:	Assign a unique identifier to the problem report.
A-1.10:	Record the date and time when the problem entered the problem report repository.
A-1.11:	Assign the status 'New' to the problem report.
<u>Report Quality Control</u>	
A-2.1:	Study the problem report.
A-2.2:	Check the correctness, consistency and completeness of the reported data.
A-2.3:	Contact the problem submitter in cases when the reported data are not correct, consistent, or complete.
A-2.4:	Confirm the problem. (At this step, the support engineer only checks whether the problem report communicates a problem)
A-2.5:	Revise and complement the description of the software problem, if necessary.
A-2.6:	Check whether the problem is unique.
A-2.7:	Assign the support engineer's own judgement of problem of priority.
A-2.8:	Assign the problem report to the relevant support engineer - <i>Service Team Member</i> .
A-2.9:	Record the additional activities/tasks performed during the phase 'Report Quality Control'.
A-2.10:	Assign the status 'Opened' to the report.
<u>Problem Investigation</u>	
A-3.1:	Study the problem report.
A-3.2:	Check the correctness, consistency and completeness of the reported data.
A-3.3:	Confirm the problem.
A-3.3.1:	If one suspects that the reported problem is unique, do the following:
A-3.3.1.1:	Study the software system.
A-3.3.1.2:	Define a set of test cases required for problem investigation.
A-3.3.1.3:	Execute the software system until the problem is recreated.
A-3.3.1.4:	Revise and complement the description of the software problem, if necessary.
A-3.3.2:	If one suspects that the reported problem is a duplicate, do the following:
A-3.3.2.1:	Update the master problem report with additional information on the problem found in the duplicate problem report, if any.
A-3.3.2.2:	Record the Report ID of the unique problem report (in text).
A-3.3.2.3:	Close the management of the duplicate software problem report.
A-3.3.3.4:	Assign status "Withdrawn" to the problem report.
A-3.3.4:	Record the results of problem investigation.
A-3.3.5:	Suggest solutions to the problem, if any.
A-3.3.6:	Investigate whether there is a any solution to temporarily work-around the problem.
A-3.3.7:	Revise the maintainer's judgement of the problem priority.
A-3.3.8:	Submit additional problems encountered during the activity 'Problem Investigation'.
A-3.3.9:	Record the additional activities/tasks performed during 'Problem Investigation'.
A-3.3.10:	Record the effort and resources of the activity 'Problem Investigation'.
A-3.3.11:	Assign the status "Cancelled" to the duplicate problem report.
A-3.3.12:	Notify the back-end process about the problem report.
A-3.3.13:	Notify the customer about the status of the problem resolution, if necessary.
<u>Problem Supervision</u>	<u>Solution Delivery</u>
A-3.3:	Conduct system testing
	A-4.1: Deliver the release containing the problem solution to the customers affected by the problem.
	A-4.2: Announce to other customers that the release containing the problem solution is ready for delivery.

problem was detected. It is also important to store the date when the problem was encountered/discovered. We do not need to store this data for every software product. We need this data for only safety-critical systems and the systems in which time is a critical performance factor. Its value in combination with the severity value, enables us to assess and predict the reliability of the software system. It may also greatly facilitate problem investigation, and help reproduce or analyse problems that are keyed to the particular time of day such as workloads.

The date of reporting the problem is just as important. It helps in defining the problem age and in tracking problem resolution over time. This, in turn, provides insight into the effectiveness of the corrective maintenance process.

Due to the nature of their products (administrative products), CGE&Y does not regularly record the temporal aspects of the problem occurrence. However, if this piece

of information is relevant and pivotal for problem investigation, it is then stored together with the textual *Problem Description* field during the CGE&Y Activity A.1.4 in Table 2. Concerning the problem reporting date, CGE&Y's tool automatically records this date during the CGE&Y Activity A-1.10 in Table 2.

Mapping Criteria 4: Product and its environment is recorded (CM³: Process Activities:A-1.6-A-1.8 in Table 1): If a maintenance organisation maintains more than one product that is being used by several customers, and if these customers possess different releases of the products, it is then imperative that the product and its release be identified and reported [4]. Release identifier specifies the version of a product containing the problem. This information should then help the maintenance engineer recreate the problem in the identified version.

Usually, a product is divided into a number of functional areas. A functional area corresponds to some

part of a product executing a clearly identified function. The identification of these areas allows for a rough localisation of a problem.

The product-environment relation describes the connection between the product and the elements of its environment. It is essential for the maintenance personnel to know the relation of the product to its environment. This knowledge can be used to predict how changes in these elements will affect the software product and vice versa. This also helps better understand the problem.

CGE&Y automatically records product identification data like product ID, release ID and environment. The customer only needs to identify the name of the product affected by the reported problem (see CGE&Y Activity 1.5 in Table 2). This is because the CGE&Y process studied is dedicated to one and only one customer. CGE&Y has full access to the system and its environment as utilised by their customer. Hence, they have control of the product, its version, and the product environment. Regarding the identification of the software components affected by the problem, CGE&Y does not have any fields reserved for identifying the affected components. This information may be stored together with the textual *Problem Description* field during the CGE&Y Activity A-1.4, if relevant (see Table 2).

Mapping Criteria 5: Problem Submitter is recorded (CM³: Process Activities: A-1.9 in Table 1): The problem submitter must be identified. If the submitter is not identified, there is a great risk that this problem will not get resolved, since further information from the submitter may be needed for problem resolution. Or, if resolved, it will not be easy to deliver the solution to the right customer. This may then have a negative effect on customer satisfaction. Non-identification of a customer may obstruct the problem resolution process. Often, the engineers at the support process level (see Table 1) have a need to contact the customers when questions arise [1, 5]. Hence, they should have an easy access to the submitter identification data.

Concerning the CGE&Y process and its supporting tool, it is entirely dedicated to only one customer. Hence, the customer organisation is always identified. Even the persons and additional contact persons from the customer organisation are identified, in cases when need arises for discussing the problem and its solutions (see CGE&Y Activities: A-1.1 and A-1.7 in Table 2).

Mapping Criteria 6: Internal and External Problem Reports are distinguished (CM³: Process Activities: A-1.10 in Table 1): It is important to distinguish between internal and external problem reports. This enables priority assignment to software problems, and aids in evaluating the quality of a software product from the perspective of a customer, and aids in assessing the effectiveness of quality assurance and quality control procedures.

CGE&Y distinguishes between internal and external problem reports. This is automatically done by the tool during the CGE&Y Activity A-1.1 in Table 2. Right now this field however, is not used for any process and quality analysis and control purposes.

4.2 Problem Analysis

Our mapping criterion for the “*Problem Analysis*” phase is the following:

Mapping Criteria 7: A problem description is studied, analysed, and confirmed (CM³ Activities: A-2.1.1.1 – A-2.1.1.3, A-2.1.2.1 – A-2.1.2.3, together with all of its sub-activities in Table 1). To be able to confirm the existence of a problem, the support organisation must study and analyse the reported problem. Not paying enough attention to the problem description, the support engineer may fail in recognising whether the reported problem is really a problem and/or whether it is unique or duplicate. The consequence is that the support engineer escalates the reported problem to the next support level (either the next front-end support level or the back-end support level). This in turn implies disruption interfering in actual maintenance work on the next support process level.

Duplicate problem reports are a serious concern to maintenance organisations. They create a large amount of extra work such as repeated data collection, repeated reporting, and repeated diagnosing of the same problem, distorted statistics of the maintenance effort and of the quality of a product, and wasted service resources [8, 9, 10]. Only a careful investigation may reveal the presence of a software problem.

At CGE&Y, all reported software problems first undergo a careful study and analysis. Their contents is checked for completeness, consistency, and correctness (see CGE&Y Activities A-2.1.1 – A-2.1.2, A-2.2.1 – A-2.2.2 in Table 2). If the reports are incomplete, inconsistent or incorrect, CGE&Y contacts the customer for complementary additions (see CGE&Y Activity A-2.1.3 in Table 2). Afterwards, all problems get confirmed. The level of confirmation depends on the phase. During the “*Report Quality Control*” phase, one only checks whether the report really communicates a software problem (see CGE&Y Activity A-2.1.4 in Table 2). During the “*Problem Investigation*” phase, one executes software till the problem shows itself (see CGE&Y Activity A-2.2.3 together with all its sub-activities in Table 2).

4.3 Problem Supervision

We have defined the following two mapping criteria for the “*Problem Supervision*” phase:

Mapping Criteria 8: The front-end support organisation continuously supervises the progress of the management of the problem at the next front-end

or back-end support process level (CM³: Activity: A-2.2.1 – A-2.2.2 in Table 1). To expiate a demand may take time, especially when it concerns a software problem. Meanwhile, customers may require information on the progress of its resolution. To satisfy this, the front-end support organisation should provide their customers with status information of the problem resolution and/or temporary work-arounds, if any. The reported status concerns the progress made at both the next front-end and back-end support levels.

CGE&Y does not supervise the management of problem reports at the back-end support level. As soon as the problem report gets escalated to the back-end process level, the CGE&Y is relieved of the ownership responsibilities for the problem. This means that the back-end process level takes over the responsibilities for managing the communication with the customer. However, due to the fact that they have access to the back-end process via their tool, the front-end support may follow the problem management there, if need arises.

Mapping Criteria 9: The front-end support takes part in evaluating the suggestion for modification design from the customer and support perspective (CM³: Activity: A-2.2.2 and its sub-activities in Table 1). It is not always enough that suggestions for changes and plans for their realization are approved by the engineers' at the back-end process level. There must be an enterprise-wide authority for approving the appropriate suggestion for change and for approving a plan for its realization. Such a group is usually called a *Change Control Board (CCB)*. Its members should represent various roles within an enterprise who have interest in deciding upon the change. Some of these members should come from the front-end support. Their role is to examine the appropriateness and correctness of the change from the customer and front-end support perspective.

In business situations with many users, it may be difficult to determine and evaluate the effect of changes made to the product. Being the most optimal from the maintenance perspective, many suggestions for changes may have a substantial impact on the customer and/or support operation. Front-end support engineers have the best knowledge of the customers, their business, usage profiles, and technology literacy and friendliness. Providing a daily support to their customers, they are also able to recognise the changes which may negatively affect the performance of the front-end support process.

At CGE&Y, the CCB meetings are not relevant. As already mentioned in Section 2, the process was specifically developed for one particular customer and product. In addition, changes made to the product usually impact a limited number of users. Due to the fact that the CGE&Y department studied tightly co-operates with the customer and its users, this impact is easy to communicate, evaluate and assess.

Mapping Criteria 10: The front-end support participates in system testing (CM³: Activity: A-2.2.3 and its sub-activities in Table 1). The front-end support engineers know the system best. They are usually very few ones within the whole IT enterprise who have the complete “big picture” of the system, and knowledge of how it is operated on by the users. Hence, they are a highly valuable resource within system testing, during which, extensions, adaptations, and corrections are being tested. When investigating the newly reported software problems, they must also conduct testing in order to confirm the existence of a software problem.

At CGE&Y, the support engineers take part in some of the system tests (see CGE&Y Activity A-3.1 in Table 2). They do it in cases when they possess competency for doing the type of the system tests required.

5. Final Remarks

CM³: *Upfront Maintenance* is a process model developed at ABB in the years 2000-2001. It was developed by studying two front-end real-life industrial support processes at two ABB organisations. In this paper, we evaluated it in the context of one small department belonging to Cap Gemini Ernst & Young (CGE&Y). This department performs a process especially tailored to one and only one specific customer and product. Other departments at CGE&Y perform a standard process globally defined for the whole organisation.

Our results show that most of the CGE&Y front-end activities could be easily mapped on CM³ activities. We have however observed some discrepancies. These discrepancies are mainly due to the fact that the CGE&Y process was mainly dedicated to one specific product and one customer and it was run by one small department. Hence, the CGE&Y process does not have any need for activities like identifying the product, its ID, and environment. This data is already recorded in their process supporting tool. Another discrepancy concerns the activities during the “*Problem Supervision*” phase. Due to the fact that the front-end support engineers at CGE&Y are relieved of the ownership responsibility when escalating the problem report to the back-end support level, they do not supervise any problem management there. Finally, the CCB-meetings are not relevant in the context of a process tailored to manage only one customer and one product.

6. Epilogue

Today, researchers possess very little insight into many industrial processes. One of the least visible processes is the front-end support. In this paper, we have attempted to provide the software community with some insight into the upfront maintenance activities. We did this by presenting the activities inherent in the CM³: *Upfront Maintenance*

process model and by mapping them on one of the CGE&Y upfront maintenance processes. The mapping process may be considered as one little validation step of *CM³: Upfront Maintenance*. More mapping and validation work however needs to be done in order to determine its industrial credibility.

Acknowledgement

First of all, we would like to thank *The Swedish Research Council* (in Swedish *Vetenskapsrådet*) for their financial support of the project *Software Maintenance Laboratory: Upfront Maintenance*. This study has been made possible thanks to their recognition of the importance of this area.

We would also like to thank the manager and support manager at CGE&Y for helping us understand their upfront maintenance process. They are Per Tidén, the manager who initiated our co-operation with CGE&Y and Åsa Gavelin, the support manager who helped us study the *Problem Handling* process at CGE&Y.

References

- [1] Arthur L J, *Software Evolution: The Software Maintenance Challenge*, John Wiley & Sons, 1988.
- [2] Bouman J, Trienekens J, van der Zwan M, Specification of Service Level Agreements, clarifying concepts on the basis of practical research, Proceedings of Software Technology and Engineering Practice, STEP '99. Sept. 1999, pp. 169-178.
- [3] Glass R L, Noiseux R A, *Software Maintenance Guidebook*, Prentice-Hall, 1981.
- [4] Hazeyama A, Hanawa M, *A Problem Report Management System for Software Maintenance*, In Proceedings, IEEE International Conference on Systems, Man and Cybernetics, 1999, Vol. 1, pp. 1019-1024.
- [5] Kajko-Mattsson M, *Corrective Maintenance Maturity Model: Problem Management*, PhD thesis, ISBN Nr 91-7265-311-6, ISSN 1101-8526, ISRN SU-KTH/DSV/R--01/15, Department of Computer and Systems Sciences (DSV), Stockholm University and Royal Institute of Technology, 2001.
- [6] Kajko-Mattsson, M., Tjerngren, L.-O., Andersson, A., *CM³: Upfront Maintenance*, in Proceedings, Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute, 2001.
- [7] Kajko-Mattsson M, *Infrastructures of Virtual IT Enterprises*, IEEE International Conference on Software Maintenance, IEEE Computer Society Press: Los Alamitos, CA, 2003, pp. 199-208.
- [8] Layzell P J, Macaulay L, *An Investigation into Software Maintenance-Perception and Practices*, In Proceedings, IEEE International Conference on Software Maintenance, 1990, pp. 130-140.
- [9] Lee I, Pitt G, Iyer R K, *Efficient Service of Rediscovered Software Problems*, In Proceedings, IEEE Annual Symposium of Fault Tolerant Computing, 1996, pp. 348-352.
- [10] Wedde K J, Stalhane T, Nordbo I, *A Case Study of a Maintenance Support System*, In Proceedings, IEEE International Conference on Software Maintenance, 1995, pp. 32-41.

Mapping UML Diagrams to a Petri Net Notation for System Simulation

Zhaoxia Hu and Sol M. Shatz
Concurrent Software Systems Laboratory
University of Illinois at Chicago
{zhu, shatz}@cs.uic.edu

Abstract. *UML statecharts are widely used to specify the dynamic behaviours of systems. To support systematic simulation of such models, we propose an approach to map systems specified using UML diagrams to colored Petri net notations. Simulation results are provided in form of self-defined trace files and Message Sequence Charts. A prototype tool is described. One unique feature of our research is the support for user-controlled view of the system simulation.*

1. Introduction

The Object Management Group (OMG) adopted a new paradigm for software development called Model Driven Architecture (MDA) [1] to recognize the fact that models are important artifacts of software development and they serve as a basis for systems as they evolve from requirements through implementation.

In this paper, we propose a UML-CPN transformation framework to introduce dynamic model analysis into UML modeling [2, 3] by mapping UML models to Petri net models, in particular colored Petri nets (CPNs) [4]. This work is aimed at investigating model-driven simulation. In general, simulation can be used to create scenarios based on design models. Evaluation of the scenarios generated by simulation runs helps to reveal potential design errors in an early stage of system development. To leverage on existing techniques and tools, and to formalize UML object models and their interrelationships, we let a net model serve as the engine that drives the simulation. In our framework, statechart diagrams and collaboration diagrams are adopted as our primary notation for modeling behavior. Statechart diagrams are first converted to colored net models; the UML collaboration diagrams are then used to guide the connection of these object models, providing a single CPN for the system under study.

To simulate UML statechart models, a number of commercial statechart simulators are available, such as

Rhapsody [5] and ObjectGEODE [6]. Our interest is to investigate techniques that are useful, but not adopted or only weakly adopted, by other tools. We present flexible visualization of scenarios generated through simulation runs to provide users customized views of simulation at different levels of abstraction. In particular, for this paper we investigate techniques for providing meaningful Message Sequence Charts (MSC) [7] that provide views of system simulation to users to help them understand the system itself as well as its properties. We demonstrate techniques for model-driven view control of simulation traces, e.g., selecting some of the model components or a subset of the events.

A prototype tool has been developed to support the automated generation of colored Petri net models from UML notations and to provide users with an interface to control visualizations of the simulation result.

2. Background

We first provide a very brief introduction to statecharts and colored Petri nets.

UML statecharts UML statecharts are an object-based variant of classical (Harel) statecharts [8]. In this paper, we deal with a subset of UML statecharts for the purpose of focusing on the general approach of generating net models from UML specifications and exploring techniques for model simulation. The incremental feature of our approach supports future inclusion of other components of statecharts, such as composite states.

Colored Petri nets Petri nets are a mathematically precise model, and so both the structure and the behaviour of Petri net models can be described using mathematical concepts. We assume that the reader has some familiarity with basic Petri net modeling [9]. Petri nets can be “executed” to perform model analysis and verification. Colored Petri nets (CPNs) [4] are one type of Petri net. In colored Petri nets, tokens are differentiated by *colors*, which are data types. Places are typed by *colorsets*, which specify which type of tokens can be deposited into a certain place. Arcs are associated with inscriptions, which are expressions defined with data values, variables, and functions. Arc inscriptions are used to specify the enabling condition of the associated transition as well as the tokens that are to be consumed or generated by the transition.

This material is based upon work supported by the U.S. Army Research Office under grant number DAAD 19-01-1-0672, and the U.S. National Science Foundation under grant number CCR-9988168.

3. Overview of the UML-CPN architecture

The architecture of the UML-CPN approach is depicted in Fig. 1. The UML-CPN approach integrates three types of application software that are represented by three rectangles in Fig. 1. The Rational Rose tool supports UML modeling, and the Design/CPN tool [10] supports modeling in colored Petri nets. In our framework, we use the Design/CPN tool as the underlying simulation engine. Thus, our approach leverages upon existing theory and tools. The UML-CPN conversion tool is our prototype tool. A typical run of the transformation approach is as follows. The Rational Rose tool is used to design a UML model specified with statecharts and collaboration diagrams. The conversion tool converts the UML model into a CPN model that can then be loaded into the Design/CPN tool for simulation. We call the generated colored Petri net notation supported by Design/CPN the *target model*. The simulation results are presented to the user in two formats: Message Sequence Charts (MSCs) and simulation traces. The simulation traces have a text format and contain complete information regarding the system simulation, while the MSCs can be tailored by the user according to his/her interests regarding the system under study. As we will discuss later, the conversion tool provides the user with an interface to control the views of system behaviour so that only the information that the user is most interested in is displayed in the MSCs.

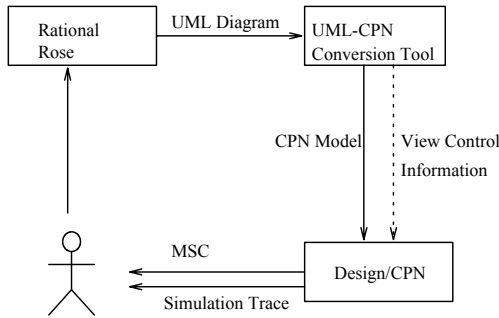


Figure 1. The architecture of the approach

The generation of the target model is based on an abstract net model that was previously presented in [11]. The purpose of defining the abstract model is to be consistent with the ideas of MDA and achieve separation of concerns. Therefore, the transformation of UML specification to colored Petri net notation is divided into two steps. First, a UML model is converted to an abstract net model. In the second step, the abstract model is enriched with the syntax supported by Design/CPN to generate the target model – platform specific model (PSM), in the MDA terminology.

Before we introduce the abstract net model, let us have an overview of the basic mapping between the constructs of UML statecharts and those of colored Petri nets. A statechart consists of states and transitions labelled with events and actions. A Petri net model consists of

places, transitions, arcs and tokens. Naturally, the transformation from a statechart to a Petri net is accomplished by the following mappings: a state is mapped to a place; a transition is mapped to a Petri net transition and a set of arcs; and events and actions are mapped to tokens. The concept of “events” is a key factor in defining the execution semantics. In fact the actions of creating, routing, and dispatching of events primarily determine the execution semantics of state machines. Since an event is modeled by a token in the CPN model, we will use *event-tokens* to refer to the tokens derived from events of statecharts from now on.

3.1. Abstract CPN models

An abstract *system-level* model consists of Object Net Models (ONMs) and an Internal Linking Place (ILP) place. An ONM, derived from a UML statechart, describes the behaviour of an individual object and defines the token routing mechanism within an object. The ILP place defines the communication between the objects.

We start by introducing the structure of Object Net Models (ONMs), as shown in Fig. 2, and described in detail in [11]. An ONM consists of a lifetime behaviour model (LM) and a token routing structure. LM represents an abstract colored Petri net that is derived from the statechart of an object and describes the object’s lifetime behaviour, as defined by the state changes captured in the object’s statechart diagram. As shown in Fig. 2, three places – *input place* (IP), *output place* (OP), and *event router place* (ER) – and two transitions – T1 and T2 – defined the token routing structure for an object. The *input place* of the object holds the event-tokens that will be used by the object. The *output place* of the object holds the event-tokens that will be routed to other objects. The *event router place* holds the event-tokens that are generated by the object. When the object generates an event-token, the token can have a type of either *external* or *internal*. As shown in Fig. 2, the input place, IP, is connected to LM, indicating that IP holds the event-tokens that will be consumed by the object. Thus, for each transition internal to LM, if this transition is a triggered transition, there will be an input arc originating from the place IP. Likewise, the place ER is connected “from” LM because ER holds the tokens that are generated by the object.

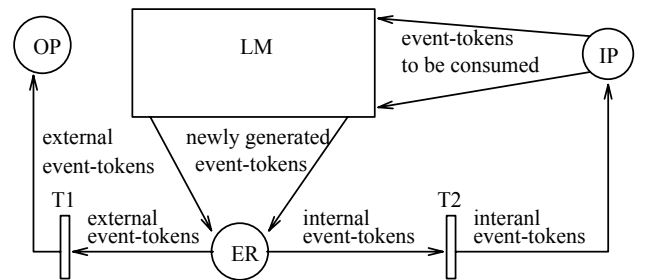


Figure 2. The structure of an Object Net Model

Recall that a state machine is described in terms of a hypothetical machine that has three key components: an event queue, an event processor, and an event dispatcher mechanism [3]. In Fig. 2, the place *IP* holds the event queue. *LM* represents the event processor. The behaviour of the event dispatcher mechanism is handled by the non-deterministic feature of transition firings inherent in Petri net models.

In order to construct a system-level model, the inter-object communications must be integrated into the model. To simplify our approach, we assume the existence of a simple collaboration diagram that defines the event flows between the objects. In the abstract model [11], we defined a special place, an *Internal Linking Place (ILP)* that is used to route event-tokens between the object models.

4. The target model

In order to explore model-driven simulation using an existing simulation engine, we transform the abstract model into a target model, suitable for direct use in the tool called Design/CPN.

4.1. Target model structure

The basic structure for a system-level target model is shown in Fig. 3. A target model consists of modules; in this case, the modules are called pages. Four types of pages are defined for target models: object page, *INL* (Internal Net Linkage) page, main page, and *Init* page. Recall that a system-level abstract model consists of ONMs (one model per object) and an *ILP* place. Accordingly, in the target model, each ONM is represented by an object page that defines the object behaviour described initially with a statechart in the UML notation; and the *ILP* place is represented by an *INL* page that holds the part of the Petri net that is responsible for inter-object communication, which is captured by collaboration diagrams in UML. The net structure of the target model consists of a two-level tree structure and one initialization page, called the *Init* page. The top level of the tree structure is the main page, which contains the high level constructs of the system. The bottom level of the tree structure contains the object pages and the *INL* page that defines the detailed constructs of the system. For convenience, we call object pages and the *INL* page *subpages*. The tree structure alone defines an executable system-level model. Moreover, we define an *Init* page for initializing the net model.

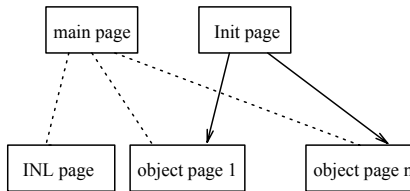


Figure 3. The structure of a target model

We will explain the target model via a simple example. Fig. 4 shows the UML model of a Master-Servant (MS) system where two objects (*Master* and *Servant*) interact with each other. The UML model consists of two statecharts and one collaboration diagram. Initially, the *Master* object is in the state *Init*. When the transition labelled with */Start*, which is a trigger-less transition, fires, a new event *Start* is generated and the *Master* object enters the state *Waiting*. The *Servant* object is initially in the state *Idle*. When event *Start* occurs, the transition labelled with *Start* fires, and the object enters the state *Active*. Fig. 4 (c) shows the simple collaboration diagram that depicts the event flow between the two objects.

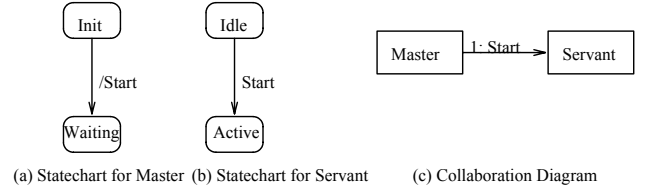


Figure 4. The UML model for the MS system

4.2. Main page

The main page depicts a high-level view of the model structure with the help of substitution transitions. A *substitution transition* represents a *subpage* of the net structure (The effect is the same as if the page that the transition represents appeared physically at the site of the transition). A *subpage* is connected to the main page via special places, which are called *sockets* and *ports*. A *socket* is a place defined on the main page while a *port* is a place defined on a subpage. A socket and a port constitute a pair and they are conceptually the same place. A socket can be associated to multiple ports to connect multiple subpages to the main page. When the model is executed, tokens are allowed to be exchanged through the socket and its associated ports. Fig. 5 shows the main page of the MS example.

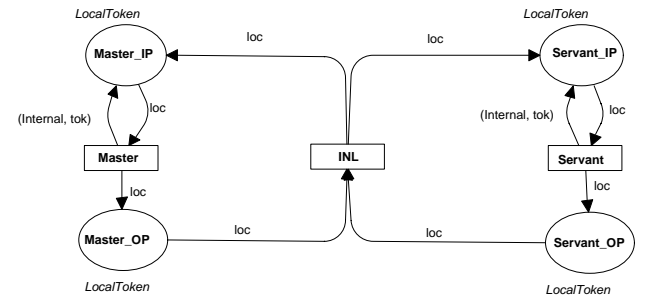


Figure 5. The main page of the MS system

The main page contains three *substitution transitions* that represent, respectively, the *Master* object page, the

Servant object page, and the *INL* page. These subpages represent the bottom level of the tree structure. In Design/CPN, a place is drawn as an ellipse. The four places in Fig. 5 serve as sockets in the model. The sockets connect to the ports in the *INL* page and object pages to “glue” the pages into a system-level net. The postfix “IP” and “OP” of the place names stand for input and output places, respectively. One thing worth noting is that a place has a *colorset* that is a data type that determines the type of tokens that the place is allowed to contain, as we mentioned in Section 2. The four places in the main page have colorset *LocalToken*. Instances of *LocalToken* are event-tokens that we previously introduced in Section 3.

4.3. Implementing ONMs as object pages

An Object Net Model (ONM) describes the behaviour of an object. We refine these models into the target model by mapping each ONM to an object page. Naturally, the structure of an object page captures both the behaviour modeling and the token routing that characterize any ONM.

In order to understand the model behaviour, it is necessary to have in mind the definitions for tokens. For an object page, we define two types of tokens. One type of token represents the activeness of a *state-place*. A *state-place* is a place that is derived from a state of the statechart. We call this type of token *active token*. An active token is denoted as *A*. If a state-place holds an active token, this place is active, denoting that the object is in the state modeled by this place. The other type of token is *event-token* that we introduced previously in Section 3. An event-token can be either an external or internal token, denoted as (External, event_name) or (Internal, event_name), respectively. Fig. 6 shows the object page for the *Servant* object of the MS example.

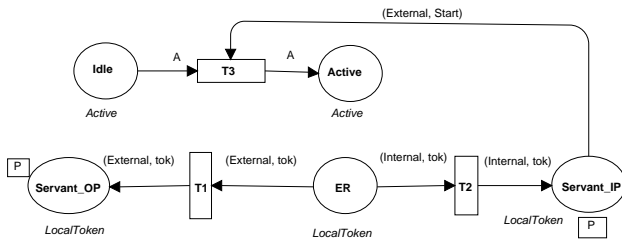


Figure 6. The object page for the Servant object

In Fig. 6, the object behaviour is modeled by the state-places *Idle* and *Active*, and transition *T3*. The other components of the object page implement the token routing structure. The input place *Servant_IP* and the output place *Servant_OP* place serve as two ports through which the object page is connected to the main page. A port place can be identified by a small rectangular, labelled with a

letter *P*. In Fig. 6, assume that the *Idle* place holds an active token, denoting that the object is in the *Idle* state. When an event-token, denoted as (External, Start), arrives in place *Servant_IP*, transition *T3* is enabled. When *T3* fires, the event-token is consumed, modeling that the event *Start* is dispatched by the event dispatcher of the state machine, and an *active token* is deposited into the place *Active*.

When a UML transition fires, an action can be performed. This type of case is modeled by generating an event-token corresponding to the action. Accordingly, an arc is then added from the corresponding transition to the *ER* (event router) place. Whenever an event-token is created, a type attribute is attached to the token. The attribute can be *Internal* or *External* – specifying if the event is to be responded to by the same object that creates the event (i.e., the creator object) or if the event is to be sent to other objects.

Due to lack of space, details on the *INL* and *Init* pages are not provided.

5. Simulation traces and visualization

Design/CPN provides a generic facility to save simulation reports, but the automatically generated reports are not straightforward in terms of providing an end-user with domain-specific information. So, we extend the idea by generating self-defined traces by using *code segments* as supported by Design/CPN. A *code segment* is a sequential piece of code that is defined for a Petri net transition and executed each time the transition occurs. We define code segments for recording the following information to a simulation trace: the object, source states, target states, the triggering event, and newly generated event. Fig. 8 (a) shows a very simple example simulation trace of the MS system. It records the history for firing two transitions.

Another method of observing simulation results is by using Message Sequence Charts [7], which have an intuitive graphical appearance and capture the message passing between the objects of a system. An MSC shows a history of events in terms of a timeline for each object. We define code segments that invoke an MSC library [12] to create the visual MSC diagrams. The MSC library provides two functions, call them *c1* and *c2*, for generating components of MSCs. Function *c1*, which has two parameters – an object and a label – generates a solid square on the timeline associated with the object. The label-parameter is placed next to the square icon. We invoke function *c1* with an object-parameter *obj* and a label-parameter *event_name* to visualize that a triggering event, *event_name*, is consumed by object *obj*. Function *c2*, which has three parameters – a sender object, a receiver object, and a label – generates a labelled arrow originating from the timeline of the sender object and targeting the timeline of the receiver object. We invoke function *c2* with the parameters *sender_obj*, *receiver_obj*, and *event_name*, to visualize that an event with name

event_name is sent from object *sender_obj* to object *receiver_obj*.

The target model contains two primary classes of net transitions – those that directly correspond to *UML transitions* and those that support the modeling of *UML transitions*. It is the transitions in the first category that are critical for generating Message Sequence Charts. We call these transitions *critical transitions*. For example, transition *T3* in Fig. 6 is a critical transition. Transition *T3* is directly derived from the transition of the statechart in Fig. 4 (b). On the other hand, transitions *T1* and *T2* belong to the second category of net transitions; they are support transitions. In this case, *T1* and *T2* are defined for routing event-tokens.

The main components of Message Sequence Charts are those that visualize the message passing between objects. The following procedure outlines the basic steps for defining the code segments to generate these components with respect to some object model, *obj*.

Notations

- *CT(obj)*: The set of critical transitions associated with the target model for object *obj*.
- *consume(t, e)*: A Boolean condition that evaluates to true if transition *t* is specified to consume an event-token *e*.
- *generate(t, receiver_obj, e)*: A Boolean condition that evaluates to true if transition *t* is specified to generate an event-token *e* that can be consumed by a critical transition associated with object *receiver_obj*.

```

foreach t ∈ CT(obj) do
  if consume(t, e1) and generate(t, receiver_obj, e2)
  then generate a code segment that contains the
    following two function calls:
      c1(obj, e1);
      c2(obj, receiver_obj, e2);
  else if consume(t, e1)
  then generate a code segment that contains the
    following function call:
      c1(obj, e1);
  else if generate(t, receiver_obj, e2)
  then generate a code segment that contains
    the following function call:
      c2(obj, receiver_obj, e2);
  endif
endif
endif
enddo

```

When a critical transition fires the associated code segment is executed, which invokes the MSC library functions. Thus, the following information is captured in an MSC: consuming of the triggering event, and sending of newly generated events (if there are any) to destination objects. For instance, consider Fig. 8 (b). The arrow represents that the *Master* object generates a new event called *Start* and sends this new event to the *Servant* object.

The solid square represents that the *Servant* object has received the event *Start* and this event triggers a transition.

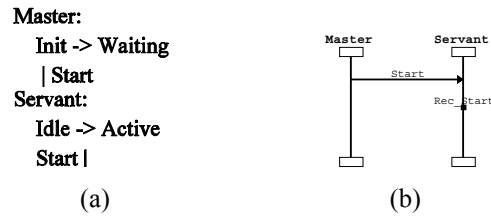


Figure 8. A simulation trace and MSC for MS system

5.1. User-controlled views of system simulation

A complex distributed system may consist of many objects that communicate with each other through message passing. As a means to control the complexity of systems analysis, designers view systems at different levels of abstraction. To aid this process, we want a designer to be able to reason about the behaviour of a subset of the objects or the occurrences of some particular events. Accordingly, an MSC can be defined to capture the behaviour of a subset of the objects and/or the occurrences of some selected events. In this section, we introduce the idea of filters to tailor the views for the system behaviour. We have defined two types of filters: *object filters* and *event filters*. Object filtering allows the user to select objects of interest – those objects whose behaviours are to be captured in the MSCs; and event filtering allows the user to constrain the information displayed in the MSCs by selecting events of interest. These two types of filters can be used together to control the views of system behaviour. Our prototype tool provides interfaces to allow the user to choose different views of the system behaviour using these filters.

The key idea for implementing the view control is that the target model is parameterized based on the selected objects and events so that the model can produce different views during the simulation, according to the user's choices. The view control technique can help a user check for the following types of properties (among others) in simulation traces: A given event occurs; an event does not occur; an event *P* is followed/preceded by another event *Q* (we will illustrate this last one in our example).

6. An example

We illustrate our approach by a small example of a gas station system. This example is adapted from [13]. The system consists of a customer and a pump that processes the customer's request for filling gas. The customer must prepay for the gas. After the customer pays, (s)he is allowed to select the gas grade and then press the nozzle. Then the pump starts to fill gas into the tank. The pump stops filling when the prepaid money is spent or the tank is full. If the customer's prepaid amount is more than the value of the gas filled into the customer's gas tank, the

pump will prompt the customer to pick up the change. The customer can change his/her decision and cancel the request for filling the tank after (s)he pays or selects the gas grade. After the customer cancels the request, the pump will return the customer's prepaid money. A simple UML model for this example consists of two statecharts and one collaboration diagram. Statecharts for the *Customer* object and *Pump* object are shown in Figures 9-10¹. As we will discuss shortly, these models contain some error to be revealed by our simulation analysis.

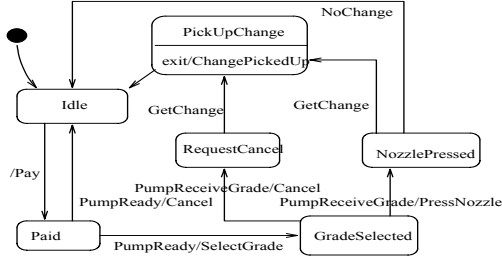


Figure 9. Statechart for Customer

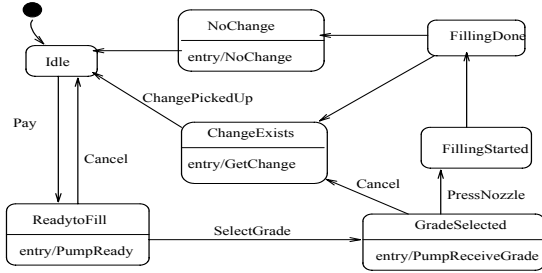


Figure 10. Statechart for Pump

For illustration, let us suppose that we want to check the following property:

Once the customer cancels the request for purchasing gas, the customer's prepaid amount should be returned.

This property can be checked by inspecting the MSCs generated from simulation runs. Fig. 11 (a) shows an MSC generated by a simulation run using our tool. We can see from the figure that many messages have been passed between these two objects. Since we are particularly interested in two events, *Cancel* and *GetChange*, we use event filtering to help us remove unwanted information

¹ In the statecharts, there exist transitions that originate from the same source state and have the same triggering event (or, both transitions do not have a triggering event). This is being done to model a pure nondeterministic choice.

from MSCs. We use the interface provided by our prototype tool to select the two events of interest. Now, the MSC generated from a simulation run is shown in Fig. 11 (b). We can clearly identify that none of the first three occurrences of *Cancel* is followed by an occurrence of *GetChange*, indicating an error in terms of the desired behaviour of the source model. Although this is a very simple example, it illustrates the intended purpose of our model-driven view control. Due to lack of space, we let the reader determine how to modify the source statechart models.

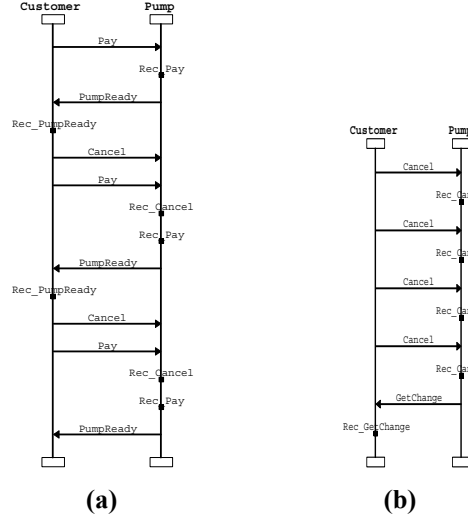


Figure 11. Message Sequence Charts for Gas Station

7. Related work

There are other research efforts that concentrate on supporting validation and analysis of UML statecharts by mapping UML diagrams to other formal target notations [14]. One such research effort aimed at validating UML models was the work on a tool called vUML [15]. This work used the information contained in the class diagrams, statechart diagrams and collaboration diagrams of a model to generate the Promela specification. The SPIN model checker was used to perform the verification. In [16], the authors present a branching time model-checking approach to the automatic verification of correctness of UML statecharts. In this work, statecharts are first translated into hierarchical automata to provide a formal semantic basis for verification. In our work, we adopt Petri nets as the target language since Petri nets have strength in their graphical notations and a mature theoretical base. Furthermore, as highlighted in this paper, we have found that net models can be flexibly adapted to drive simulation experiments.

Work with a similar motivation as ours includes [17]. Our work differs from this work in that we hide the underlying simulation engine from the end user by presenting the simulation results using the constructs from

the original UML model. In addition, our transformation approach is automated. The work of Dong et. al. [18] is closely related to our work. They presented an approach of using Hierarchical Predicate Transition Nets (HPrTNs) to define and integrate UML statechart diagrams and collaboration diagrams. Thus, these efforts are quite complementary to our basic approach. However, the HPrTN model is defined in a more abstract manner than our CPN model. Also, our research extends the basic translation technique to create a model that can be imported into an existing CPN support tool and to use this tool for investigating model-driven simulation.

Other research efforts that relate to statechart simulation include those presented in [19, 20]. One idea that separates our work from other statechart simulation research is that we introduce view control into the simulation process to help a user understand and reason about the behaviour of a system.

8. Conclusions and future directions

In order to explore the validation and verification of UML models through transformation, we follow a transformation framework to structure UML models as colored Petri net models. We enrich the abstract net model and present a pragmatic CPN model so that the resulting model can be imported into Design/CPN for simulation and analysis. The transformation is based upon the execution semantics of state machines. To help users to facilitate the simulation features supported by Design/CPN, we derive Message Sequence Charts to visualize simulation results. A prototype tool has been developed to support the automated generation of executable CPN models from UML notations. One unique feature of the tool, which is facilitated by the net construction, is the support for interfaces that enable a user to control the visualization of system simulations. We provide object filters as well as event filters to control the views for the system behaviour.

One direction for future work can be to extend our methodology so that it supports the transformation of more complex features of UML statecharts, such as concurrent composite states. Another direction is to investigate the integration of other UML diagrams in our approach to strengthen the behavioural modeling and analysis.

Acknowledgments

We thank C. Xiong for her help in development of the prototype tool. We also thank the reviewers for their

helpful suggestions that improved the presentation of the work.

References

- [1] OMG. Model Driven Architecture, www.omg.org/mda.
- [2] G. Booch, I. Jacobson and J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
- [3] OMG. UML semantics 1.5, 2003. www.cgi.omg.org/uml.
- [4] L.M. Kristensen, S. Christensen, K. Jensen, "The Practitioner's Guide to Coloured Petri Nets," *International Journal on Software Tools for Technology Transfer*, 1998, Springer Verlag, pp. 98-132.
- [5] E. Gery, D. Harel, and E. Palatshy. "Rhapsody: A Complete Lifecycle Model-Based Development System." In *Proc. Int'l Conf. on Integrated Formal Methods*, 2002. pp. 1-10.
- [6] ObjectGEODE. www.csverilog.com.
- [7] ITU-T Recommendation Z.120: Message Sequence Chart. International Telecommunication Union; Telecommunication Standardization Sector (ITU-T), 1999.
- [8] D. Harel. "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, vol. 8, 1987, pp. 231-274.
- [9] T. Murata, "Petri Nets: Properties Analysis and Applications." In *Proc. the IEEE*, 77(4), April 1989, pp. 541-580.
- [10] Design/CPN, www.daimi.aau.dk/designCPN/.
- [11] J. A. Saldhana, S. M. Shatz, and Z. Hu, "Formalization of Object Behavior and Interactions From UML Models." *Int'l Journal of Software Eng. and Knowledge Eng.*, 11(6), 2001, pp. 643-673.
- [12] MSC Library, www.daimi.au.dk/designCPN/libs/mscharts/
- [13] Gas station example, sunset.usc.edu/classes/cs599_2000/GasStation.pdf
- [14] J. Whittle. "Formal Approaches to Systems Analysis Using UML: An Overview." *Journal of Database Management*, 11(4), 2000, pp. 4-13.
- [15] J. Lilius and I. Paltor, "vUML: A Tool for Verifying UML Models." In *Proc. the 14th Int'l Conf. on Automated Software Engineering*, IEEE. 1999, pp. 255-258.
- [16] S. Gnesi, D. Latella, and M. Massink. "Model checking UML statechart diagrams using JACK" In *Proc. Int'l Symp. on High Assurance Systems Eng.* IEEE. 1999, pp. 46-55.
- [17] R. G. Petti IV and H. Goma, "Improving the Reliability of Concurrent Object-Oriented Software Designs." In *Proc. the 9th Int'l Workshop on Object-oriented Real-time and Dependable Systems (WORDS)*, October, 2003.
- [18] Z. Dong, Y. Fu, and X. He. "Deriving Hierarchical Predicate/Transition Nets from Statechart Diagrams." In *Proc. Software Eng. and Knowledge Eng.*, 2003, pp. 150-157
- [19] A. Egyed and D. Wile, "Statechart Simulator for Modeling Architectural Dynamics." In *Proc. the 2nd IEEE/IFIP Conf. on Software Architecture (WICSA)*, 2001, pp. 87-96.
- [20] I. Ober, S. Graf, and I. Ober, "Validating Timed UML Models by Simulation and Verification." *UML 2003 Workshop - Specification and Validation of UML Models for Real Time and Embedded Systems*, 2003.

Multi-Objective Optimization by CBR GA-Optimizer for Module-Order Modeling

Taghi M. Khoshgoftaar, Yudong Xiao, and Kehan Gao
Florida Atlantic University, Boca Raton, Florida USA

Abstract

In the case when resources allocated for software quality improvement are limited or unknown, an estimation of the relative rank-order of modules based on a quality factor such as number of faults is of practical importance to the software quality assurance team. This is because improvements can be targeted toward a set of most faulty modules according to resource availability. A module-order model (MOM) can be used to determine the relative rank-order of modules. A MOM usually ranks the modules according to the predicted number of faults obtained from an underlying quantitative prediction technique, such as multiple linear regression and case-based reasoning. In this paper we propose a computational intelligence-based method for targeting the performance behavior of MOM(s). The method maximizes the number of faults accounted for by the given percentage of modules enhanced. A new modeling tool called CBR GA-optimizer is developed through a synergy of genetic algorithms (GA) and case-based reasoning (CBR). The tool automatically finds the best CBR fault prediction models according to a project-specific objective function.

1 Introduction

Software quality classification models that identify software modules as fault-prone and not fault-prone [1, 6, 8, 11] have been used to direct enhancement resources toward the low-quality modules. The degree of software quality improvement efforts is dependent on the availability of reliability enhancement resources. Software quality classification models require that the individual quality-based groups be defined prior to modeling, usually via a threshold on the

number of faults expected. However, in software engineering practice the software management team often cannot choose an appropriate quality threshold at the time of modeling. Therefore, a prediction of the rank-order of modules from the most to the least faulty is more practical. Based on such a predicted rank-order, the software quality management team can target a set of the most faulty modules for enhancement as per the available resources.

A module-order model (MOM) is a technique that estimates the rank-order of modules according to a quantitative quality factor, such as number of faults. A MOM is constructed through an underlying quantitative prediction model, such as multiple linear regression [7] and case-based reasoning [8]. Previous works associated with MOM mainly concentrate on how to improve the accuracy of the quantitative prediction model by minimizing the average, relative, or mean square errors [7]. However, it is the predicted rankings of program modules that affect the behavior of MOM(s) and not the predicted value of the quality factor. In this paper, we propose a method that directly targets the performance behavior of MOM(s). More specifically, for a given number of modules enhanced, we are interested in maximizing the number of faults accounted for by the prediction models. A genetic algorithm is ideally suited (in conjunction to an underlying prediction model) for such a direct optimization. Existing software quality prediction techniques, such as multiple linear regression and case-based reasoning, cannot achieve such a direct optimization, because the optimization objective is often highly discontinuous with multiple minima or maxima.

By combining genetic algorithms (GA) [2, 10, 13] with case-based reasoning (CBR), a CBR GA-optimizer tool is developed. The underlying quantitative model used is the one obtained by CBR, while the GA-optimizer automatically finds the best CBR models according to a given objective function. The developed tool can be used to solve the multi-objective optimization problem [4, 14] related to CBR. Instead of minimizing the quantitative error such as AAE for the underlying prediction model, we directly maximize the MOM performances for a given set of cutoff percentiles, i.e., the percentages of modules enhanced for the given data set.

The proposed methodology is validated through a case study of a full-scale industrial software system. Four performance goals were chosen for optimization purposes: maximizing the MOM performances for the 5%, 10%, 20%, and 30% of the total number of modules enhanced. This selection is based on our discussion with, and the inputs provided by, the software management team of the system under consideration. The four performance goals are combined into an objective function by empirically determining the appropriate weights. To demonstrate justification of using a sophisticated software quality modeling method such as the one proposed, we compared the MOM performances based on the GA-CBR technique with those based on ordering by the lines of code (LOC) metric. It is shown that the proposed GA CBR-based MOM generally had better performances as compared to ordering by LOC. To our knowledge, this is the first study to use GA to implement a performance optimization for building MOM(s) based on CBR.

The remainder of this paper continues with the next section, which presents the details of the module-order modeling technique. This is followed by a section that presents the case-based reasoning and CBR GA-optimizer modeling methods, and a section that discusses our case study of a wireless configuration software system. Finally, the conclusion and suggestions for future work are presented in the last section.

2 Module-Order Modeling

A module-order model (MOM) is used to predict the relative rank-order, and hence software quality, of each program module based on a set of product and process metrics. The primary advantage of using a MOM over a software quality classification model is that it enables project managers to enhance as many modules beginning with the most faulty in the rank list as the available resources allow. Usually, a MOM is calibrated according to the following three steps: (1) Build an underlying quantitative software quality prediction model, such as a software fault prediction model; (2) Rank the program modules according to a quality measure predicted by the underlying model; (3) Evaluate the accuracy of the predicted ranking.

Initially, a quantitative software quality prediction model is calibrated to predict the dependent variable, which in our studies is the number of faults associated with a program module. The software fault prediction modeling technique used in our study is CBR. For a given quantitative model, the number of faults, F_i , in module i is a function ($F_i = f(\mathbf{x}_i)$) of its software measurements, the vector \mathbf{x}_i . Let $\hat{F}(\mathbf{x}_i)$ be the estimate of F_i by a fitted model, $\hat{f}(\mathbf{x}_i)$. In module-order modeling, the predicted values of the dependent variable obtained by $\hat{F}(\mathbf{x}_i)$ are only used to obtain the (predicted) relative order of each program module.

Let R_i be the percentile rank of observation i in a perfect ranking of modules according to F_i . Let $\hat{R}(\mathbf{x}_i)$ be the percentile rank of observation i in the predicted ranking according to $\hat{F}(\mathbf{x}_i)$. The following steps illustrate the evaluation procedure for a module-order model. Given a prediction model and a data set having modules indexed by i :

1. Management will choose to enhance modules in a priority order, beginning with the most faulty. Determine a range of percentiles that covers management's options for the last module that will be enhanced, based on the schedule and resources allocated for reliability enhancement. Choose a set of representative cutoff percentiles, c , from that range.
2. For each c , determine the number of faults accounted for by modules above the percentile c . This is done for both the perfect and predicted ranking of the modules: $G(c)$ is the number of faults accounted for by the modules that are ranked (perfect ranking) above the percentile c , and $\hat{G}(c)$ is the number of faults accounted for by the modules that are predicted as falling above the percentile c . Therefore, a higher value of c corresponds to the more faulty modules.

$$G(c) = \sum_{i: R_i \geq c} F_i \quad (1)$$

$$\hat{G}(c) = \sum_{i: \hat{R}(\mathbf{x}_i) \geq c} F_i \quad (2)$$

3. Calculate the performance of the MOM, $\phi(c)$, which indicates how closely the faults accounted for by the model ranking match with those of the perfect module ranking.

$$\phi(c) = \frac{\hat{G}(c)}{G(c)} \quad (3)$$

where $\phi(c)$ is a number between 0 and 1. It is desired that $\phi(c)$ be close to 100% (or 1) for the c of interest.

4. After evaluating the performance of a MOM, it is ready for use on a currently underdevelopment similar project or subsequent release. Determine the predicted ranking, by ordering modules in the current data set according to $\hat{F}(\mathbf{x}_i)$, and subsequently, compute the respective $\phi(c)$ values.

3 Case-Based Reasoning

A CBR system [9] attempts to find a solution to a new problem based on previous experiences, represented by a *case-base* or *case library*. A solution algorithm uses a *similarity function* to measure the relationship between the new

problem and each case in the case-base, and finally retrieves relevant case(s) and determines a solution to the new problem. A CBR system, therefore, consists of three major components: a case-base, a similarity function, and a solution algorithm. Information related to the past cases is stored in a case-base, which is often the training data set. A case is composed of a set of independent variables and a dependent variable, which in our study is the number of faults. Using the cases in the case-base, a model is trained and is then applied to a test data set, which contains information related to program modules of a similar project. In order to retrieve the relevant case(s) in the case-base that are most similar to the new problem, a similarity function is used.

A similarity function measures the distance between the new problem and all the cases in the case-base. Modules with the smallest distances are considered similar and designated as the *nearest neighbors* (N) [3]. The commonly used similarity functions include: City Block, Euclidean, and Mahalanobis distances [8]. We use the latter, because it explicitly accounts for correlation among the independent variables. Let d_{ij} be the distance from the new case (or module) under investigation, \mathbf{x}_i , to each of the cases in the case-base, \mathbf{c}_j . The Mahalanobis distance is given by:

$$d_{ij} = (\mathbf{c}_j - \mathbf{x}_i)' S^{-1} (\mathbf{c}_j - \mathbf{x}_i) \quad (4)$$

where prime (') represents a transpose, S is the variance-covariance matrix of the independent variables over the entire case-base, and S^{-1} is its inverse.

By using a solution algorithm, we can estimate the number of faults of the new case under investigation. Let n_N be the number of the nearest neighbors that are to be used to obtain the solution to the new problem. The prediction of the dependent variable (number of faults) of the target module, $\hat{F}(\mathbf{x}_i)$, can be calculated by a weighted average of dependent variables accounted for by the n_N nearest neighbors. In this case study, an *inverse-distance weight* was used in a weighted average. Since a smaller distance implies a better match, we weight each case in the nearest neighbors set, N , by a normalized inverse distance, $\delta_{ij} = \frac{1/d_{ij}}{\sum_{j \in N} 1/d_{ij}}$. The prediction of the dependent variable of the target module i is then given by,

$$\hat{F}(\mathbf{x}_i) = \sum_{j \in N} \delta_{ij} F_j \quad (5)$$

For the given similarity function and solution algorithm used by our CBR-based fault prediction model, the number of the nearest neighbors is the only adjustable modeling parameter.

4 CBR GA-Optimizer

The problem of finding the best model in a CBR system can be considered an optimization problem. Generally

speaking, optimization refers to finding the best solution(s) in some specific search space Ω , according to a given objective function F_{obj} . Because the function $F_{obj} : \Omega \rightarrow \mathcal{R}$ (where \mathcal{R} is a set of real numbers) is usually discontinuous and the search space Ω may be very large, no traditional mathematical methods can be used to solve such an optimization problem. Genetic algorithm (GA) offers an interesting and natural approach to solve such a problem [5].

GA starts from a set of initial solutions, and uses biologically inspired evolution mechanisms to derive new and possibly better solutions. It starts from an initial population P_0 , and generates a sequence of populations P_1, \dots, P_n , by using three types of operations within the population: *crossover*, *mutation*, and *reproduction*. The elements of the population are called *chromosomes* and the fitness of each chromosome is measured by a *fitness function*. Each chromosome consists of a set of *genes*. For each generation, the algorithm selects some of chromosomes and uses the crossover (for pairs), mutation (for singles), or reproduction operations on them, with some given probabilities, respectively. *Crossover* mixes genes and *mutation* randomly changes some genes. Each pair of chromosomes creates a new pair. Each generation inherits some chromosomes from the last generation and accepts some newly created chromosomes according to a given probability. The fitter chromosomes have a greater chance to be inherited into the next generation. The algorithm stops when a certain criterion is satisfied or a pre-defined number of generations is reached.

We developed a new tool, named "CBR GA-optimizer", by using a GA-engine that searches for the best models yielded by the CBR-solver. The CBR GA-optimizer consists of two major components: GA-engine and CBR-solver. The GA-engine creates the population of the chromosomes and implements the evolution process as described above. It sends each chromosome to the CBR-solver and receives the objective function value from the CBR-solver as a feedback for the chromosome evaluation. The CBR-solver receives a chromosome from the GA-engine, carries out the complete CBR process and builds a CBR model. The CBR model calculates the objective function value and sends it back to the GA-engine.

For a given system, finding the best performance, i.e., $\phi(c)$, of a MOM for a set of cutoff percentiles of interest is a multi-objective optimization issue. In our case study, maximizing the $\phi(c)$ values at the 95%, 90%, 80% and 70% percentiles was desired. These four performance goals are combined to obtain the objective function that is to be optimized by the CBR GA-optimizer. The objective function is given by,

$$F_{obj} = w_{.95} * \phi(.95) + w_{.9} * \phi(.9) + w_{.8} * \phi(.8) + w_{.7} * \phi(.7), \quad (6)$$

where $w_{.95}$, $w_{.9}$, $w_{.8}$, and $w_{.7}$ represent the weights of the performances at the respective cutoff percentiles of interest.

The weights can be determined according to the importance of each individual objective in the context of the optimization problem under consideration.

The general form of the optimization issue related to MOM can be presented as follows: for a given fit data set, test data set, similarity function, and objective function F_{obj} , find some solution(s), p , in the search space, $\Omega = \{n_N\}$, that maximize $p \in \Omega F_{obj}(p)$. As mentioned earlier, in the context of this paper the number of the nearest neighbors is the only parameter that can affect the performance of the underlying CBR-based quantitative prediction model. Therefore, the search space includes only one parameter, n_N . By using the CBR GA-optimizer, the GA-engine automatically searches for the best model created by the CBR-solver, according to the optimization problem shown in objective function (6).

5 Empirical Case Study

5.1 System Description

This case study (denoted as WLTS) involves data collection efforts from initial releases of two large Windows®-based embedded systems used primarily for customizing the configuration of wireless telecommunications products. The two C++ applications provide similar functionalities, and contain common source code. Hence, both systems are studied simultaneously. The main difference between them is the type of wireless product that each supports. Both systems consist of over 1400 source code files, and each system contained more than 27 million lines of code. Software metrics were obtained by observing the configuration management systems and the problem reporting systems of the applications. The problem reporting system tracked and recorded problem statuses. The fault data represents the faults discovered during system tests. Upon preprocessing and cleaning the collected data, i.e., removal of incomplete data points, 1211 modules remained. Over 66% of modules (809) were observed to have no faults, and the remaining 402 modules had at least one or more faults. An impartial data splitting was performed on the data set in order to obtain the fit (807 modules) and test (404 modules) data sets. To avoid biased results due to a lucky data split, the original data set was randomly split 3 times to obtain 3 pairs of the fit and test data sets. However, due to space considerations we only present the results for one data split, i.e., Split 1.

The five software metrics used for reliability modeling for this case study are:- *BLOC*: the number of lines of code for the source file version prior to the coding phase, i.e., auto-generated code; *SLOC*: the number of lines of code for the source file version delivered to system tests; *B.COM*: the number of lines of commented code for source file version prior to coding phase, i.e., auto-generated code;

S.COM: the number of lines of commented code for source file; and *INSP*: the number of times the source file was inspected prior to system tests. The collection and use of these metrics for modeling purposes were dependent on their availability and the available data collection tools. The product metrics indicate the number of lines of source code prior to the coding phase and just before system tests. The inspection metric, *INSP*, was obtained from the problem reporting systems of the two embedded applications.

5.2 CBR GA-Optimizer Methodology

In the GA-engine, some parameters associated with GA were set as follows: (1) Reproduction rate = 0.5; (2) Crossover probability = 0.9; (3) Mutation probability = 0.08; (4) Number of generations = 3000; (5) Size of population = 200; (6) Number of runs = 50. The optimization of GA parameters is beyond the scope of this study, however, is part of our future research. At the end of each run, the two best models are selected. Hence, at the end of all the 50 runs there were 100 candidate models among which we selected the best model, i.e., the one with the highest value for objective function (6).

In the CBR-solver, an n -fold cross-validation (also commonly known as the leave-one-out technique) was implemented on the fit data set to train the underlying quantitative (fault) prediction model. It is an iterative process such that during each iteration, one of the n observations in the fit data set is used as the test data and the other $n - 1$ are used to train or build the model. The Mahalanobis distance was used as the similarity function and the inverse-distance weighted average was used as the solution algorithm. The GA-optimizer engine initially creates a genome, i.e., number of nearest neighbors, n_N , and sends it to the CBR-solver to build a corresponding module-order model. The CBR-solver returns the value of the multi-objective function to the GA-optimizer engine. Then, the evolution process starts until the termination condition is met. The GA-optimizer finally outputs some “best” MOM(s), which maximizes the objective function (6), which consists of a weighted sum of performances at the cutoff percentiles of interest, i.e., $\phi(.95)$, $\phi(.90)$, $\phi(.80)$, and $\phi(.70)$.

In the context of the multi-objective optimization for MOM(s), one of the key issues is to assign the suitable weights to the objective function (6). From a practical software engineering point of view, it is beneficial and cost-effective to begin reliability enhancements with the most faulty modules. Since a higher cutoff percentile value corresponds to the more faulty modules, we assigned the highest weight to the performance at $c = 95\%$. Subsequently, we assigned decreasing weights to the performances at the $c = 90\%$, 80% , and 70% percentiles. We considered ten different sets of weights for the performances at the four c val-

ues. The weights for each set were such that $w_{.95} > w_{.90} > w_{.80} > w_{.70}$. Subsequent to analysis based on each weight set, it was observed that performance of the models were not impacted by the different weight sets. The results presented in this paper are based on $w_{.95} = 10$, $w_{.90} = 6$, $w_{.80} = 3$, and $w_{.70} = 1$. As a future work, we shall consider using the GA-optimizer for directly optimizing the weights for each performance goal.

5.3 Results and Analysis

5.3.1 MOM Calibrated by CBR GA-Optimizer

The performance of the best model for Split 1 is shown ¹ in Table 1. The models were built using the five software metrics described earlier. The first column of the table lists the cutoff percentiles from 95% through 65% with decrements of 5%. We present the MOM performance beyond the lowest cutoff percentile of interest to the management team. The table shows the performances for both fit and test data sets. For each cutoff percentile, the $G(c)$, $\hat{G}(c)$, and $\phi(c)$ values are presented. For a given c , the number of faults is influenced by the way the original data set is split into the fit and test data sets. For example, the total numbers of faults in the fit and test data sets for Split 1 are 1071 and 786 faults. As shown in last row of the table, these values represent the respective number of faults accounted for at $c = 65\%$. This implies that for Split 1, a $c = 65\%$ for a perfect ranking will account for 100% of faults in the fit and test data sets. This was also observed with the other two splits.

We observe that for the cutoff percentiles of interest, although the performances at the higher percentiles in the multi-objective function were assigned larger weights, the final performances on the higher percentiles were not close to the respective objectives. This may be reflective of: (1) the underlying prediction model used for obtaining the predicted rankings of modules, and (2) the characteristics of the software metrics data [12]. The latter, is especially reflected in the software quality modeling of high assurance systems (such as WLTS), in which the percentage of faulty modules is usually a very small fraction of the total number of modules. In the case of Split 1, the performance of MOM on the test data set is generally better than that on the fit data set with the exception at $c = 95\%$. This was also observed for Split 3. However, for Split 2, the performance of MOM on the test data set was similar to that on the fit data set except for the cutoff percentiles from 90% to 80%. This implies that the performance of MOM is impacted by the way the original data set is randomly split into the fit and test data sets.

¹The models for the other two splits are not shown due to space considerations, however; similar empirical results were obtained.

Table 1. Performances of Split 1

c	Fit			Test		
	$G(c)$	$\hat{G}(c)$	$\phi(c)$	$G(c)$	$\hat{G}(c)$	$\phi(c)$
0.95	558	487	87.28 %	508	406	79.92%
0.90	734	630	85.83 %	609	551	90.48%
0.85	859	734	85.45 %	680	625	91.91%
0.80	946	807	85.31 %	731	670	91.66%
0.75	998	877	87.88 %	760	711	93.55%
0.70	1038	932	89.79 %	780	730	93.59%
0.65	1071	973	90.85 %	786	743	94.53%

Table 2. Comparisons of Performances for Split 1 Test

c	$G(c)$	$SLOC$	GA-CBR	Difference
0.95	508	82.68 %	79.92 %	-2.76 %
0.90	609	81.28 %	90.48 %	9.20 %
0.85	680	86.18 %	91.91 %	5.74 %
0.80	731	85.77 %	91.66 %	5.88 %
0.75	760	84.34 %	93.55 %	9.21 %
0.70	780	82.31 %	93.59 %	11.28 %
0.65	786	81.93 %	94.53 %	12.60 %

5.3.2 Comparison with a Simple Method

The software quality assurance team of a given software project is often interested in knowing how well a given software quality model performs as compared to obtaining a model based on a simple rule of thumb, such as software size. This is often needed from a practical point of view for justifying the use of a sophisticated method such as the CBR GA-optimizer. In order to evaluate the MOM built by the CBR GA-optimizer, we compare it with the performance obtained when the modules are ordered according to their LOC. The LOC is often used as a heuristic practice to detect and enhance problematic software modules. This comparison is only done for the test data sets, because the generalization performance of a software quality model is of more interest to a practitioner. In the case of the ranking based on LOC, the modules in the test data are ranked according to their LOC prior to the system release, i.e., $SLOC$. The subsequent performance calculation is done using the procedure described earlier.

The comparison between the CBR GA-optimizer model and performance obtained by LOC-based ranking (notations LOC and $SLOC$ are used interchangeably to imply an ordering based on lines-of-code) is shown in Table 2, which shows the performances for Split 1. It is observed that when the top 10% ($c = 90\%$) modules of the test data set are chosen for reliability enhancements, then the MOM calibrated by the CBR GA-optimizer will have over 90% effectiveness, in other words, this MOM can detect over 90% faults ac-

counted by the top 10% of the most faulty modules according to the perfect rank-order, i.e., $609 \times 90.48\% = 551$ faults. In contrast, the ordering of modules based on *SLOC* shows an 81% effectiveness, which implies that the *SLOC*-based model can only detect about $609 \times 81.28\% = 495$ faults. The difference of the effectiveness (for $c = 90\%$) between the two approaches is about 9.20% (56 faults) in the favor of the CBR GA-optimizer.

The performance, $\phi(c)$, of the MOM calibrated by the CBR GA-optimizer generally increased as more (predicted) faulty modules are considered for reliability enhancements, i.e., as c decreases. However, this trend was not observed for the LOC-based method, i.e., the performance fluctuates (increases or decreases) as more numbers of predicted faulty modules are subjected to reliability enhancements. Overall, the MOM(s) calibrated by the CBR GA-optimizer have better performance accuracy when $c < 90\%$ as compared to those of the LOC-based rankings. In addition, the performance generally gets better with a decrease in the c value. In our experiments with the other two splits, the empirical results and conclusions were similar to those presented for Split 1.

6 Conclusion

A module-order model is a very attractive software quality improvement technique that can predict the rank-order of modules according to a quantitative quality factor, such as number of faults. Previous works associated with MOM try to improve the prediction accuracy of MOM through minimizing some quantitative error measure. However, minimizing such errors does not have a direct relationship to improving the performance of a MOM. In contrast, we presented a different approach in that we directly optimize the performance behavior of a MOM: We incorporate the desired performance goals of a MOM into an objective function which is optimized.

The proposed method combines GA with CBR, forming the CBR GA-optimizer. The underlying quantitative prediction model is CBR, and the GA-engine is used to find the best solutions to the predefined objective function. Four performance goals were selected to form the objective function based on discussions with the management team. An empirical case study of a wireless configuration system was used to validate our proposed methodology. It was shown that the GA CBR-based MOM had at least 80% effectiveness in detecting the faulty modules.

A comparison of the proposed method is done with the simple method based on ordering by lines of code. The results show that our proposed MOM has a better performance as compared with the LOC-based approach for $c < 90\%$. Further improvement of the GA CBR-based MOM performance is largely restricted by the underlying fault predic-

tion model (CBR) itself. However, the proposed methodology itself provides a novel multi-objective optimization approach. Future work will include further empirical validation via case studies of other software systems. Other prediction methods such as neural networks may also be used in conjunction with the GA-optimizer for the stated multi-objective module-order modeling problem.

References

- [1] L. C. Briand, W. L. Melo, and J. Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering*, 28(7):706–720, July 2002.
- [2] C. J. Burgess and M. Lefley. Can genetic programming improve software effort estimation? a comparative evaluation. *Information and Software Technology*, 43(14):863–873, December 2001.
- [3] B. Dasarthy. Nearest neighbor norms: NN pattern classification techniques. IEEE Computer Society Press, 1991.
- [4] C. M. Fonseca and P. J. Fleming. Genetic algorithms for multiobjective optimization: Formulation discussion and generalization. In S. Forrest, editor, *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, Urbana-Champaign, IL, USA, June 1993. Morgan Kaufmann.
- [5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [6] K. E. Imam, S. Benlarbi, N. Goel, and S. N. Rai. Comparing case-based reasoning classifiers for predicting high-risk software components. *Journal of Systems and Software*, 55(3):301–320, 2001.
- [7] T. M. Khoshgoftaar and E. B. Allen. A comparative study of ordering and classification of fault-prone software modules. *Empirical Software Engineering*, 4(2):159–186, June 1999.
- [8] T. M. Khoshgoftaar and N. Seliya. Analogy-based practical classification rules for software quality estimation. *Empirical Software Engineering*, 8(4):325–350, December 2003.
- [9] D. B. Leake, editor. *Case-Based Reasoning: Experience, Lessons, and Future Directions*. MIT Press, Cambridge, MA USA, 1996.
- [10] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin, 3rd edition, 1996.
- [11] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.
- [12] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering*, 27(11):1014–1022, November 2001.
- [13] A. H. Wright. Genetic algorithms for real parameter optimization. In G. J. Rawlins, editor, *Foundations of genetic algorithms*, pages 205–218. Morgan Kaufmann, San Mateo, CA, 1991.
- [14] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation*, 3(4):257–271, November 1999.

Noise Elimination with Ensemble-Classifier Filtering: A Case-Study in Software Quality Engineering

Taghi M. Khoshgoftaar and Vedang H. Joshi
Department of Computer Science and Engineering
Florida Atlantic University
777 Glades Road, Boca Raton, FL 33431
taghi@cse.fau.edu and vjoshi@fau.edu

Abstract

This paper presents a noise handling technique that attempts to improve the quality of datasets for classification purposes by eliminating instances that are likely to be noise. Our approach uses twenty five different classification techniques to create an ensemble filter for eliminating likely noise in a real-world software measurement dataset. Using a relatively large number of base-level classifiers in the ensemble filter facilitates in achieving the desired level of noise removal conservativeness with several possible levels of filtering. It also provides a higher degree of confidence in the noise elimination procedure as the results are less likely to get influenced by (possible) inappropriate learning bias of a few algorithms with twenty five base-level classifiers than with relatively smaller number of base-level classifiers. An empirical case study of a high assurance software project demonstrates the effectiveness of our noise elimination approach by the significant improvement achieved in classification accuracies at various levels of filtering.

1 Introduction

A software quality classification model aims to generalize the concepts learnt from a given dataset of software measurements and fault-proneness data with the aim of accurately classifying unseen software modules. Such a model can assist software quality improvement efforts by identifying software program modules that are likely to be *fault-prone (fp)* during operations. This facilitates cost-effective utilization of resources allocated for software test-

ing, inspection, and quality enhancement. Software measurements are key in developing a software quality estimation model because of the software engineering assumption that they hold the underlying information regarding software product quality.

The predictive accuracy of a given classification technique is influenced by two major factors: (1) Quality of the training data, and (2) Appropriateness of the chosen algorithm for the given data. Poor-quality (noisy) data, when used during training can have undesirable consequences, and hence, using appropriate noise handling procedure as a preamble to any data mining task is of paramount importance. Noisy instances in a poor-quality dataset may have either erroneous attribute values (attribute noise) or corrupted class labels (class noise). However, since machine learning algorithms usually treat noisy examples as being mislabeled [10], we feel that the noise identified by our approach could actually be either attribute noise or class noise.

The problem of effectively dealing with data noise can be approached mainly in three different ways. To cope with noise, one can either use robust (noise-tolerant) algorithms [11], try to correct noisy instances [25], or filter out noisy instances from the dataset [2].

The empirical study presented in this paper investigates the use of a noise elimination procedure in the context of software quality classification. Noise elimination with ensemble-classifier was deemed appropriate for our study. In our ensemble-classifier study, the basic assumption is that if a large number of classifiers misclassify a given software module, then it is likely that it is a noisy instance in the dataset. More specifically, such a software module suggests that its software measurements and quality data do not adhere to (or represent) the underlying characteristics of the quality of the software product. The size of the dataset was not compellingly small to choose *polishing* [25] over *noise elimination* [2]. There are many similarities between our approach to noise handling and the approach employed

in [2]. But there are some major differences too. In essence, one can say that our study leverages the work done by [2] in order to effectively handle the noise.

To our knowledge, this work is one of the few studies that examines the effect of a noise handling technique on a real-world dataset with potential inherent noise. Many empirical investigations [2, 25, 27] have evaluated different noise handling mechanisms on datasets in which noise is artificially injected, either in the class label or in the attribute values. In such cases, there is no way to ensure that the noise handling procedure improves the true classification accuracy. On the contrary, with our approach, noise free evaluation dataset is available because of the way noise filtering is performed. Also, in our noise elimination approach, the number of classifiers used is rather large. Different learning algorithms from different categories have been chosen as base-level classifiers to form noise filters. This enables us to use different levels of filtering to eliminate instances that are likely to be noise, and avoids results from being influenced by (possible) inappropriate bias of a few classifiers for the given dataset. In addition, it also raises the confidence level in the process of eliminating the data instances suspected of being noisy.

2 Noise Detection and Elimination

In the ensemble-classifier approach presented in [2, 3] for noise elimination, the reported results were based on ensemble-classifier models of only three and five different base-level classifiers, respectively. It may not be wise to form an opinion about an instance being noisy by considering only a small number of classifiers, because the appropriateness (or bias) of the chosen learning algorithms applied to a particular dataset also plays a significant role.

Experimenting with a rather large number of classifiers can ensure that we are reducing the probability of throwing away good data and raising the level of confidence in the identification of actual noisy instances. In our study, we used 25 different base-level classifiers from different computational categories, such as bayesian, instance-based, rule-based, decision-tree based, pattern-based, and statistical techniques, etc., for our ensemble classifier noise removal approach.

Unlike Brodley and Friedl's approach [2] that only considers majority filtering (the least conservative approach) and consensus filtering (the most conservative approach), our study examines the effects of different levels of noise filtering on the predictive accuracy of classifiers. We experimented with four different levels of filtering. In the context of the software measurement data investigated, we decided to eliminate the instances (considered as noise) misclassified by: 23 or more classifiers (the most conservative approach, i.e., misclassification by over 90% classifiers); 20

or more classifiers (misclassification by over 80% classifiers); 17 or more classifiers (misclassification by over 68% classifiers); and 13 or more classifiers (the majority filtering approach - the least conservative one). Thus, by using 25 base-level classifiers, we were able to achieve various levels of filtering (levels of conservativeness), instead of just majority and consensus filtering.

2.1 Handling Exceptions

Danyluk and Provost note that learning from noisy data is difficult, because it is hard to distinguish between instances that are noisy and instances that are exceptions to the general rule, especially if the noise is systematic [6]. Brodley and Friedl also indicated in their paper [2] that one has to be cautious not to unknowingly remove exceptions from the dataset while trying to eliminate noisy instances.

We believe that our ensemble-classifier approach, especially the approach with the most conservative level of filtering, does counteract the above problem to a certain extent. While it is true that not all the classifiers can capture the atypical nature of the instances that are exceptions to the general case, we believe that with our most conservative approach, it is likely that at least three of the twenty five classifiers would have the appropriate biases that could allow them to correctly classify exceptions. This implies that our most conservative approach, where all the instances misclassified by 23 or more classifiers are eliminated, is the least likely of all the four different levels of filtering to take exceptions for potential noise and eliminate them inadvertently.

However, it should be noted that this study was not particularly aimed at handling exceptions, and we feel that further research is necessary for better addressing this problem in conjunction with ensemble-classifier filtering.

2.2 System Description

The software metrics and quality data used in our study is that of a NASA software project, JM1, written in C++. The data was made available through the Metrics Data Program (MDP) at NASA, and included software measurement data and associated error (fault) data collected at the function/subroutine/method level. The dataset consisted of 10,883 software modules, of which 2,105 modules had errors (ranging from 1 to 26) while the remaining 8,778 modules were error-free, i.e., had no software faults. In this case study, a module with no faults was considered *nfp*, and *fp* otherwise. We note that the terms errors, defects, and faults are used interchangeably in this study.

Each module in the JM1 project was characterized by 21 software measurements: four McCabe metrics (Cyclomatic Complexity, Essen-

tial_Complexity, Design_Complexity, and Loc_Total); eight derived Halstead metrics (Halstead_Length, Halstead_Volume, Halstead_Level, Halstead_Difficulty, Halstead_Content, Halstead_Effort, Halstead_Error_Est, and Halstead_Prog_Time); four metrics of Line Count (Loc_Executable, Loc_Comment, Loc_Blank, and Loc_Code_And_Comment); four basic Halstead metrics (Unique_Operators, Unique_Operands, Total_Operators, and Total_Operands); and one metric for Branch Count. The quality of the modules was described by their *Error Rate* (i.e., number of defects in the module) and *Defect* (i.e., whether or not the module has any defects). The latter was used as the class label.

Upon removing inconsistent modules (those with identical software measurements but with different class labels) and those with missing values, the dataset was reduced from 10,883 to 8,850 modules. We denote this reduced dataset as *JM1-8850*, which now had 1,687 modules with one or more defects and 7,163 modules with no defects. We only used the 13 primitive metrics in our analysis. The eight derived Halstead metrics were not used during modeling. Thus, the classifiers were built using the 13 software metrics as independent variable and module-class as the dependent variable, i.e., fault-prone or not fault-prone.

We feel it is important to note that the software measurements used for the JM1 system were primarily governed by their availability, the internal workings of the project, and the data collection tools used by the project. The type and numbers of software metrics made available for public use was determined by the NASA Metrics Data Program. Other types of software metrics, including object-oriented measurements were not available for analysis. The use of the specific software metrics in each case study does not advocate their effectiveness – a different software project may collect and consider a different set of software measurements for analysis.

3 Experiments

We performed noise elimination using the proposed ensemble-classifier approach on the JM1 software systems data. The filtering was based on the performance of twenty five different classification techniques on the JM1-8850 dataset. For most classification techniques, the predictions on which the filtering was based were obtained using 10-fold cross-validation, with a few exceptions¹. The twenty five classification techniques used are: Case-Based Reasoning [15, 17]; Treedisc (TD) [13]; Lines-of-Code (LOC); Artificial Neural Network (ANN) [19]; Genetic Programming (GP) [20]; Rule-Based Modeling (RBM) [21]; Logistic Regression (LR) [14]; Rough Sets (RSET) [18]; Log-

¹Exceptions due to infeasibility or limitation of the tool used for RBM, TD, LR, GP, ANN, and RSET.

Table 1. Dataset Details for JM1 System

Dataset	nfp modules	fp modules	total modules	Avg. NECM $\frac{C_{II}}{C_I} = 20$
8850	7163	1687	8850	1.5359
4425-Fit	3581	844	4425	1.5691
23C-Fit	3143	753	3896	1.1457
20C-Fit	2862	696	3558	0.8384
17C-Fit	2670	646	3316	0.5786
13C-Fit	2431	576	3007	0.2724
4425-Test	3582	843	4425	1.5431
23C-Test	3143	752	3895	1.1989
20C-Test	2861	696	3557	0.8771
17C-Test	2670	646	3316	0.5904
13C-Test	2430	576	3006	0.3061

itBoost [9]; Bagging [26]; MetaCost [7]; AdaBoost [26]; Decision Table [16]; ADTrees [8]; SMO [23]; IB1 (1-NN); IBk (k-NN); PART [26]; OneR [26]; JRIP [4]; RIDOR [5]; J48 (C4.5) [24]; Naive Bayes [26]; HyperPipes [22]; and LWLStump [1]. The last 13 techniques are implemented in WEKA [26], which was used to build the respective models.

In a Lines-of-Code classifier, the modules were first sorted in an ascending order of their LOC. The underlying assumption is that a larger-size program module is likely to have more software faults than a relatively smaller-size module. Based on a specific threshold value of lines of code, thd_{LOC} , the modules with LOC lower than thd_{LOC} are predicted as not fault-prone, and the rest as fault-prone. The threshold value is varied until the desired balance between the Type I and Type II error rates is obtained.

Experimenting with as many as twenty five classifiers enabled us to explore several levels of noise filtering. We decided to have four different levels of filtering denoted by 13C, 17C, 20C, and 23C, with 13C being the least conservative and 23C being the most conservative. Noise filtering at the 13C level, a noise filtering level where all the instances misclassified by 13 or more classification techniques have been eliminated, is analogous to majority filtering since we are using twenty five classification techniques in our ensemble. We did not perform consensus filtering (25C in our case), for it appeared to be too stringent a criterion for noise elimination with twenty five classification techniques.

Having eliminated potentially noisy instances, each dataset was proportionately split into two halves: fit and test sets. The notations used for each dataset and the distribution of *fault-prone* and *not fault-prone* modules in each dataset are summarized in Table 1. Most of the notations are self explanatory. For example, 8850 stands for the original dataset (with 8850 modules) used for noise elimination; 4425-Fit and 4425-Test are respectively the training and the evaluation datasets generated before noise elimina-

tion; and 23C-Fit and Test stand for the fit and test dataset splits respectively generated after noise elimination at the 23C level. At the 23C level, 1059 (11.97%) of the 8850 modules were identified as noisy, and hence, were eliminated. Similarly, 1735 (19.60%), 2218 (25.06%), and 2837 (32.06%) of the 8850 modules in the original dataset were eliminated at 20C, 17C, and 13C levels of filtering respectively.

It was surprising to note that the distribution of the nfp (about 80%) and fp (about 20%) modules in the datasets had remained almost the same after the noise elimination process at different levels of filtering.

3.1 Expected Cost of Misclassification

Comparing the performance of different classification methods based on the two misclassification rates (False Positive - Type I and False Negative - Type II) can indeed be a difficult task, especially when the performance is being evaluated across a range of datasets (with different level of noise in our case). In order to reduce the degree of difficulty/complexity involved in the comparison task, it was decided to use ECM (Expected Cost of Misclassification) [Eq. 1], as a unified singular performance measure. In addition, unlike the overall misclassification rate performance measure, it accounts for the prior probabilities of classes and the costs of misclassifications [12]. The lower the value of ECM, the better the performance of a given classifier.

$$ECM = C_I Pr(fp|nfp)\pi_{nfp} + C_{II} Pr(nfp|fp)\pi_{fp} \quad (1)$$

where, C_I and C_{II} are costs of Type I and Type II misclassification errors respectively, π_{fp} and π_{nfp} are prior probabilities of fp modules and nfp modules, $Pr(fp|nfp)$ is the probability that a nfp module would be misclassified as fp , and $Pr(nfp|fp)$ is the probability that a fp module would be misclassified as nfp .

In practice, it is difficult to quantify the actual costs of misclassifications at the time of modeling. Normalized Expected Cost of Misclassification ($NECM = \frac{ECM}{C_I}$) [Eq. 2] avoids this problem by facilitating the use of cost ratio $\frac{C_{II}}{C_I}$, which can be more readily estimated using software engineering heuristics for the given application. In the context of the JM1 software system, the range of cost ratio values of 10 to 50 was considered practical. We investigate with cost ratio values of 10, 20, 30, and 50 for computing NECM. However, due to space limitation, results for all the cost-ratios can not be presented.

$$NECM = Pr(fp|nfp)\pi_{nfp} + \frac{C_{II}}{C_I} Pr(nfp|fp)\pi_{fp} \quad (2)$$

Table 1 shows the average value of NECM (across the 25 classifiers) at the cost-ratio of 20 (as an example). The relatively higher values of Avg. NECM for the datasets with

no noise filtering (8850 and 4425) indicates that the dataset has inherent noise, which is confirmed by the improvement in the NECM value with increasing noise filtering level, i.e., going from the most conservative level-23C to the least conservative level-13C.

3.2 Results & Analysis

As shown in the Table 2, for all the twenty five classifiers, the value of NECM tends to decrease as we go from most conservative level (23C) of filtering to the least conservative level (13C) of filtering, indicating improvement in classification accuracy.

A Two Way Randomized Complete Block Design approach was employed to investigate whether the twenty five classification techniques and the datasets with different levels of noise filtering yield significantly different NECM values with respect to one another respectively. The NECM computed for the fit and test data sets, was used as the response variable for the ANOVA models. Due to the restrictions on the paper size, only the ANOVA table for the predictive performance of the classifiers is presented in Table 3. The notations used in the table are as follows: DF - degrees of freedom, SS - sums of squares, MS - mean squares, and F - the F statistic.

Examining ANOVA results based on the predictive performance (test data) of the classifiers (Table 3) reveals that for all the cost ratios, the NECM values across the datasets (with different noise filtering levels) are significantly different – indicated by p -values less than 0.01%. Similar observation was made for the quality-of-fit results. All the classification techniques also have significantly different performances on the test datasets for the JM1 system, indicated by very low p -values (in some cases $p < 0.0001$). Similar trends were also observed for quality-of-fit performance.

The results statistically confirmed our intuitive assumption that the classification performance would improve as more and more software modules likely to be noise are eliminated. This is evidenced by the significant performance difference between the datasets with different levels of noise filtering. This was also apparent as the NECM values decreased from the most conservative level to the least conservative level of noise filtering.

A Z-test was performed to compare two different proportions – proportions of the modules identified as likely-noise by two different noise filtering approaches. First, we compared the proportion of the modules identified as noisy (and hence eliminated) by our approach (ensemble-classifier consensus filter with 25 base-level classifiers) to the proportion of the instances identified as noisy by ensemble-classifier consensus filter with only 5 base-level classifiers [3]: J48, IBk, SMO, JRIP, and LWLStump. Our consensus filter removed only 321 out of the 8850 instances,

as compared to 1425 out of 8850 instances removed by the filter with five base classifiers. When these two proportions were compared using Z-test, the computed z-value was of a very high magnitude (27.82), indicating that the two proportions are statistically different at significance level $\alpha = 1\%$. This shows that ensemble-classifier consensus filter is, statistically speaking, much more conservative with twenty five base-level classifiers than with only five classifiers.

Similarly, we also found that the ensemble-classifier consensus filter with five base classifiers is statistically more conservative than that with only three classifiers. This goes to show that as the number of base-classifiers increases, the level of conservativeness for consensus filtering increases significantly.

In the case of ensemble-classifier majority filtering for identifying potentially noisy software modules, our twenty-five classifier ensemble removed 2837 modules as compared to 2842 modules removed by a five-classifier ensemble and 2865 modules removed by a three-classifier ensemble. While there is not a markable difference in the numbers of modules removed (those that are misclassified) by the different ensemble-classifier majority filters, we note that they may not be eliminating the same modules. Among the 2837 modules identified as noise by the twenty-five ensemble-classifier, 2559 were the same modules that are identified as noise by the five ensemble-classifier. Hence, about 10% of the modules identified as noise by the twenty-five ensemble-classifier were not identified as noise by the five ensemble-classifier. Along the same lines, only 2519 modules were common between the twenty-five ensemble-classifier and the three ensemble-classifier.

4 Conclusions

The empirical study presented indicates that the predictive performance of classification techniques improves as more and more (inherent) noise is removed. Use of relatively larger number of classifiers, i.e., 25, provides certain degree of freedom and flexibility to explore different levels of filtering from most conservative to the least conservative to achieve the desired level of conservativeness while removing the instances suspect of being noisy. With twenty five base-level classifiers, it is highly unlikely for the noise elimination process to get influenced by predictions of a few classifiers which may not have appropriate inductive bias for the dataset at hand. Thus, experimenting with relatively large number of classifiers to base the noise elimination process gives a higher level of confidence in the process.

The case study presented here, a study in software measurement and software quality classification, very closely approximates a real-world scenario, where appropriate noise-handling technique(s) need to be employed on a dataset with inherent noise. The Normalized Expected Cost

Table 2. Predictive Performance for $\frac{C_H}{C_I}=20$ after Noise Elimination

Methods	13C Split	17C Split	20C Split	23C Split
CBR	0.4362	0.5202	0.8639	1.0760
TD	0.2861	0.6300	0.8791	1.2300
LR	0.3014	0.6420	0.8791	1.1974
LOC	0.3589	0.6722	0.8774	1.2483
GP	0.3234	0.6457	0.9547	1.2685
ANN	0.3194	0.6553	0.9157	1.2298
LBOOST	0.2801	0.6378	0.8679	1.1684
RBM	0.3330	0.6571	0.9536	1.2385
BAG	0.2325	0.4180	0.7186	0.9733
RSET	0.2927	0.6351	0.9185	1.2904
MCOST	0.2452	0.5920	0.7683	1.2008
ABOOST	0.2285	0.4677	0.7846	1.1019
DTABLE	0.3127	0.5250	0.7807	1.2937
ADT	0.1743	0.5359	0.9345	1.2062
SMO	0.3273	0.6601	0.9039	1.2573
IB1	0.3589	0.5449	0.9033	1.1733
IBK	0.2911	0.5259	0.8752	1.2316
PART	0.2648	0.4849	0.8544	1.2426
ONER	0.3330	0.6641	0.8327	1.1392
JRIP	0.2295	0.5084	0.9710	1.2144
RDR	0.2588	0.6408	0.8251	1.2616
J48	0.2518	0.5483	0.9103	1.0059
NBAYES	0.3619	0.6824	0.9494	1.2624
HPIPES	0.5788	0.7244	0.9843	1.2837
LWLS	0.2718	0.5413	0.8212	1.1782
Average	0.3061	0.5904	0.8771	1.1989
Std. Dev	0.0792	0.0794	0.0676	0.0837
Median	0.2927	0.6300	0.8791	1.2298
Min	0.1743	0.4180	0.7186	0.9733
Max	0.5788	0.7244	0.9843	1.2937

Table 3. Two-Way ANOVA Models for JM1 Test Datasets

$\frac{C_H}{C_I}$	Source	DF	SS	MS	F	p-value
10	Method	24	0.138	0.006	4.65	0.0000
	Dataset	4	8.384	2.096	1698.48	0.0000
	Error	96	0.119	0.001		
	Total	124	8.6402			
20	Method	24	0.483	0.020	3.00	0.0000
	Dataset	4	23.799	5.950	886.33	0.0000
	Error	96	0.644	0.007		
	Total	124	24.9272			
30	Method	24	1.053	0.044	2.60	0.0005
	Dataset	4	47.086	11.772	698.28	0.0000
	Error	96	1.618	0.017		
	Total	124	49.7578			
50	Method	24	2.865	0.119	2.33	0.0020
	Dataset	4	117.274	29.319	573.23	0.0000
	Error	96	4.910	0.051		
	Total	124	125.0495			

of Misclassification was used as a practical performance evaluation measure, taking the disparity between the two types of misclassification (very common in software quality classification and many other domains) into account. Also, the datasets on which performance of different classifiers is evaluated are *noise-free*, as they are generated by impartially splitting the given dataset after noise removal, giving a better insight into the *true* predictive performance.

While it was not the focus of our study to address the issue of exceptions, we feel that our most conservative level of filtering provides for handling exceptions to a certain degree, as it is likely that at least three out of the twenty five base-level classifiers can correctly classify the instances that are “hard-to-classify”, or are “exceptions”. We also found that there is significant difference ($p < 0.01$) in the proportion of the noise removed by consensus filtering with 25 classifiers and consensus filtering with only 5 classifiers, suggesting that consensus filtering with relatively large number of classifier is more conservative than with a few classifiers.

References

- [1] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning. *Artificial Intelligence Review*, 11(1-5):11–73, 1997.
- [2] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.
- [3] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19:45–77, 1995.
- [4] W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9-12 1995. Morgan Kaufmann.
- [5] P. Compton and R. Jansen. Knowledge in context: a strategy for expert system maintenance. In C. J. Barter and M. J. Brooks, editors, *AI'88: 2nd Australian Joint Artificial Intelligence Conference*, pages 292–306, Adelaide, Australia, November 1990. Springer.
- [6] A. Danyluk and F. Provost. Small disjuncts in action: Learning to diagnose errors in the telephone network local loop. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 81–88, Amherst, MA, 1993.
- [7] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Knowledge Discovery and Data Mining*, pages 155–164, 1999.
- [8] Y. Freund and L. Mason. The alternating decision tree learning algorithm. In *Proc. 16th International Conference on Machine Learning*, pages 124–133, Bled, Slovenia, 1999. Morgan Kaufmann, San Francisco, CA.
- [9] J. Friedman, J. Stochastic, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting, 1999.
- [10] D. Gamberger, N. Lavrač, and C. Grošelj. Experiments with noise filtering in a medical domain. In *Proc. 16th International Conf. on Machine Learning*, pages 143–151. Morgan Kaufmann, San Francisco, CA, 1999.
- [11] D. Gamberger and N. Lavrač. Conditions for occam’s razor applicability and noise elimination. In *European Conference on Machine Learning*, pages 108–123, 1997.
- [12] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, Englewood Cliffs, NJ, USA, 2nd edition, 1992.
- [13] T. M. Khoshgoftaar and E. B. Allen. Controlling overfitting in classification tree models of software quality. *Empirical Software Engineering*, 6(1):59–79, 2001.
- [14] T. M. Khoshgoftaar and E. B. Allen. Logistic regression Modeling of Software Quality. *International Journal of Reliability, Quality and Safety Engineering*, 6(4):303–317, December 1999. World Scientific Publishing.
- [15] T. M. Khoshgoftaar and N. Seliya. Analogy-based practical classification rules for software quality estimation. *Empirical Software Engineering*, 8(4):325–350, December 2003. Kluwer Academic Publishers.
- [16] R. Kohavi. The power of decision tables. In N. Lavrač and S. Wrobel, editors, *Proceedings of the European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence, pages 174–189. Springer Verlag, 1995.
- [17] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., San Mateo, California USA, 1993.
- [18] J. Komorowski, L. Polkowski, and A. Skowron. *Rough Set: A Tutorial*. Springer-Verlag, 1998.
- [19] R. P. Lippmann. An introduction to computing with neural networks. *Acoustics, Speech and Signal Processing Magazine*, 4(2):4–22, 1987.
- [20] Y. Liu and T. M. Khoshgoftaar. Genetic programming model for software quality prediction. In *6th High Assurance Systems Engineering Symp.*, pages 127–136, Boca Raton, FL, 2001.
- [21] M. Mao. Software quality classification using rule-based modeling. Master’s thesis, Florida Atlantic University, Boca Raton, Florida USA, May 2002. Advised by Taghi M. Khoshgoftaar.
- [22] J. Peng, F. Ertl, S. Bhagotra, A. Mosam, N. Vijayaratham, and I. Kanwal. Classification of U.S. census data. Data Mining Project CS4TF3.
- [23] J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report 98-14, Microsoft Research, Redmond, Washington, April 1998.
- [24] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.
- [25] C. M. Teng. Correcting noisy data. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 239–248, 1999.
- [26] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000.
- [27] X. Zhu, X. Wu, and Q. Chen. Eliminating class noise in large datasets. In *Proceedings of the Twentieth International Conference on Machine Learning*, Washington, DC, 2003.

On Modelling an e-shop Application on the Knowledge Level: e-ShopAgent Approach

Nenad Stojanovic

Institute AIFB, University of Karlsruhe, 76128 Karlsruhe, Germany

nst@aifb.uni-karlsruhe.de

Abstract. *In this paper we present an approach that models the behaviour of an on-line shop assistant on the knowledge level, i.e. it takes into account not only which actions (questions) a shop assistant will perform, but also which goals he wants to achieve by taking an action. As a generic reasoning pattern of such an e-shop agent we use the cover-and-differentiate problem-solving method, a method very successfully applied in various diagnosis and classification tasks. In that way, we can (i) model the question-answering process such that the minimal set of useful questions will be provided to a user, (ii) easily reinterpret and fine-tune shopping strategies that exist in other e-shop portals and (iii) design and integrate new methods into generic reasoning pattern. We present an evaluation study which illustrates these benefits.*

1. Introduction

A lot of effort has been spent in the last decade in replicating real-world shopping experience in e-commerce sites. Particularly, a number of models have been proposed to describe a real-world customer-buying process [1] and several recommendation strategies have been developed to represent the background knowledge and experience of a shop assistant [2]. Most of them introduce some plausible heuristics about a user's behaviour (e.g. a user should select the most preferable product among several alternatives) and in an intensive software engineering process they implement such a solution. However, the buying process can be considered as a decision-making process in which a user "searches", regarding a problem (formulated as an inquiry/query), for a solution (represented as a relevant product). Therefore, one can abstract particular e-commerce scenarios and consider the on-line shopping problem on the *knowledge level* [3]. In such a view the goal of problem solving is not

just to select one of the possible actions, but rather to construct a model of part of the world that allows the problem-solver to conclude eventually that its goals have been achieved.

In this paper we present an approach to model the behaviour of an on-line shop assistant on the knowledge level, using generic problem solving methods (PSM) [4]. Particularly, from the knowledge level point of view the problem-solving used in the e-shopping domain might be seen as a method that searches for a set of products relevant for a set of features (properties) given by a user and that refines that set (i.e. rules out some products) by introducing new features that are relevant for the user. It corresponds to the *cover and differentiate* PSM [5], very successfully applied in various diagnosis and classification tasks. In that way we model the goal an on-line shop assistant would achieve by asking a user some questions, which enables us to generate more useful questions that should be posted to a user.

This research relies on our work on interactive query refinement [6]. The corresponding system, called eShopAgent has been implemented in the KAON framework (kaon.semanticweb.org) and we set a case study that compares navigating through the same product database using a traditional and our approach.

The paper is organised as follows: in the second section we describe our approach in details. Section 3 contains a small evaluation study, whereas Section 4 contains related work. In Section 5 we give concluding remarks.

2. Modelling e-shopping problem-solving on the knowledge level

2.1 Knowledge level

The knowledge level [3] provides the means to 'rationalise' the behaviour of a system from the standpoint

of an external observer. This observer treats the system as a 'black box' but maintains that it acts 'as if' it possesses certain knowledge about the world and uses this knowledge in a perfectly rational way toward reaching its goals (*principle of rationality* - an agent will select an action that according to its knowledge leads to the achievement of one of his goals). There are three different perspectives on the knowledge level: *domain model* and *task model*, that talk in a precise and systematic way about domain knowledge and goals of the system, respectively and *problem-solving method*, that relates task and domain models in order to accomplish goals. In the meantime a lot of such generic inference patterns, called *problem-solving methods (PSM)* [47], have been identified: cover and differentiate for diagnosis [5], propose and revise [5] for parametric design, skeletal-plan-refinement for hierarchical planning etc.

2.2. E-shopping as a problem-solving

In most on-line shop systems the communication between a user and the system is initiated either by an anthropomorphic shopping agent for the given domain [8], who *transfers* his knowledge into the set of questions, or by an automatic analysis of product data, e.g. using some data-mining algorithms like ID3 [9]. The drawbacks of the first approach are well-known in the knowledge acquisition community: highly expensive hard-coding of the expert knowledge that disables its reusability in similar situations. In the second case the expert background knowledge is completely missing, such that the flexibility of the solution is lost.

Fortunately, from the knowledge level point of view the solution for an effective communication seems to be very simple: if we understand the rationale *why* a knowledge is needed, we can understand *what* knowledge should be elicited. Indeed, by analysing existing e-shop portals and their "conversations" with customers, we extracted the common behaviour (rationale) of different shop assistants, which we formulate in a simplified form like: in each action an e-shop assistant performs, he tries to eliminate as much as possible irrelevant products offered to a user. Consequently, in the elicitation process (e.g. by questioning) a shop assistant tries to acquire as much as possible "eliminating" knowledge - the knowledge that can be used for efficient elimination of products irrelevant for the current user. Finally, we can abstract this behaviour to a generic inference pattern, which (1) for a set of symptoms proposes a set of explanations and then (2) seek information to eliminate irrelevant explanations. By analysing available libraries of PSMs [47], we found a very suitable inference pattern - *cover and differentiate*

PSM (in the rest of the text abbreviated as *c&d*), developed for supporting diagnosis task [5].

2.3 cover-and-differentiate PSM

c&d is a role limiting method that implements a form of heuristic classification [10]. It resolves a problem by first proposing candidates that will cover or explain the symptoms or complaints specified by the user and then seeking information that will differentiate the candidates. The searching method is divided into a covering and a differentiate task. These tasks are abstractly defined as follows: the cover task takes a set of features (symptoms) and produces a set of candidates (explanations) that seem applicable; the differentiate task tries to rule out elements of this set. In order to achieve these goals each task uses corresponding knowledge (covering or differentiating). In Figure 1 we present the structural decomposition of the method. This *c&d* process is iterative, since some of the symptoms used to differentiate the candidates often need to be explained in a subsequent skip.

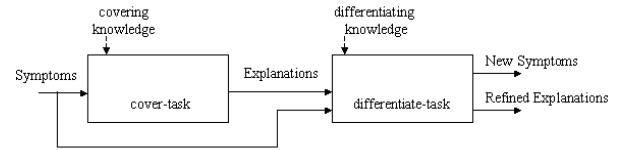


Figure 1. The structure of the *c&d* PSM

The domain knowledge used in the method should be represented as a causal network, which is the main source of the covering knowledge. More details about the *c&d* methods can be found in [5].

2.4 Using *c&d* for e-shop problem-solving

If we consider a buying scenario as the process in which a shop assistant tries to find suitable candidates (products) which satisfy (explain) a set of features a user prefers, the mapping to the *c&d* domain is straightforward: features are symptoms and products are explanations.

Therefore, from the structural point of view we can use *c&d* generic inference patterns as the problem-solving method in a shopping portal. However, the main problem is how to define covering and differentiating knowledge (relevant for *c&d*) in the e-shopping problem solving.

2.4.1 Covering Knowledge

First of all, *c&d* requires a causal network as the covering knowledge, which is not a preferred knowledge representation paradigm in the shopping domain. Therefore we post minimal requirements on the structure of the domain knowledge in an e-shop scenario and try to prepare it for the *c&d*-based processing.

The most commonly used (knowledge) structure in the e-shopping domain can be interpreted as a light weight ontology about product's features, whereas the partial orders (taxonomy) between features are explicitly specified. The corresponding knowledge base contains a set of instances for a concrete domain (a product set). On the other hand, the nature of the causality in *c&d* can be expressed as

If $cover(S, E)$ then $cover(S', E')$,

where $cover(S, E)$ means that a set of symptoms (S) can be explained with set of explanations (E), S and S' are sets of symptoms, E and E' are sets of explanations and $S' \subseteq S$ and $E \subseteq E'$. In this case we consider that symptoms $\{S \setminus S'\}$ are caused by symptoms S.

According to the *c&d* interpretation of the e-shopping scenario, this condition can be rewritten as

If $cover(F, P)$ then $cover(F', P')$,

where F, F' are sets of features and P, P' are sets of products and $F' \subseteq F$ and $P \subseteq P'$ and $cover(F, P)$ means that all products from P have all features from F. In such a causal case we consider that features $\{F \setminus F'\}$ are caused by features F.

Therefore, we need a partial order between feature-products pairs in order to "simulate" a causal network for a whole product dataset. Comparing other e-shop applications this is a very important difference – we organize products in a causal network in the first place, whereas the most of other approaches uses a decision tree topology. In [11] we gave an overview of the advantages of using causal network topology comparing to decision trees. In the rest of this subsection we show how the causality between products' features can be derived from a product dataset.

Definition 1. A product dataset can be transformed into the structure $(\Phi, \Pi, cover)$, where

- Φ is a set of all feature that exist in the given dataset. Features can be organized in the vocabulary V .
- Π is the set of all products available in the given dataset
- $cover$ is a binary relation between a set of products and a set of features, $cover \subseteq \Phi \times \Pi$. We write $cover(f, p)$, meaning that a product p has a feature f .

Definition2: A vocabulary on a set Φ of features is a structure $V := (\Phi, H)$, where H is the set of partial orders on Φ . For the relations from the H holds:

$$\forall p \in \Pi \forall f_1, f_2 \in \Phi \wedge \forall h \in H \quad cover(f_1, p) \wedge h(f_1, f_2) \rightarrow cover(f_2, p)$$

Definition 3: A product-feature node (in the rest of the text: node)

A product-feature pair is a tuple, $N = (\Phi_x, \Pi_y)$ where:

- $\Phi_x \subseteq \Phi$, Φ_x is called a set of node_features ;
- $\Pi_y \subseteq \Pi$, Π_y is called a set of node_products;

$$\Pi_y = \{ p \in \Pi \mid (\forall f \in \Phi_x) \rightarrow cover(f, p) \}.$$

In order to model causal knowledge we define two relations, equivalence and subsumption, on the set of nodes.

Definition 4: Structural equivalence ($=$):

$(\Phi_{x1}, \Pi_{y1}) = (\Phi_{x2}, \Pi_{y2}) \Leftrightarrow \Pi_{y1} = \Pi_{y2}$, which can be written as

$$N_1 = N_2 \Leftrightarrow \Pi_{y1} = \Pi_{y2} \quad (1)$$

Two nodes are structurally equivalent if their product sets are the same. Note that this relation is reflexive, symmetric and transitive.

Definition 5: Largest equivalent node (*len*) for the node $N_a = (\Phi_{xa}, \Pi_{ya})$ is a node $N_l = (\Phi_{xl}, \Pi_{yl})$ such that:

$$\Phi_{xl} = \bigcup \Phi_{xi} \mid \forall i \quad N_i = N_l \text{ and } \Pi_{yl} = \Pi_{ya} \quad (2)$$

Definition 6: Structural subsumption (parent-child):

$$(\Phi_{x1}, \Pi_{y1}) < (\Phi_{x2}, \Pi_{y2}) \Leftrightarrow \Pi_{y1} \subset \Pi_{y2}. \quad (3)$$

Two nodes are structurally subsumed if the product set of a node is subsumed by the product set of another. Note that this relation is symmetric and transitive.

For a node N_i we define the *direct child* relation as follows: $N_2 <_{dir} N_1$ iff $N_2 < N_1 \wedge \neg \exists N_i, N_2 < N_i < N_1$ (4)

In that case we call N_2 a *direct_child* of N_1 .

Finally, the partial order " $<$ " on the set of all nodes defines a lattice structure, which reflects the causality between features of products. In other words, it defines covering knowledge for *c&d* problem-solving in e-shopping domain. This covering knowledge is used in the cover-task of the problem-solving in order to find a set of relevant explanations (covering). Theoretically, the entire causal knowledge can be calculated using formal concept analysis FCA [12], since a set of all equivalent product-feature nodes corresponds to a formal concept. We omit here the mapping process. Since that FCA calculation can be very expensive and since we do not need the whole covering knowledge at once due to the iterative nature of the *c&d*, we can generate only a relevant portion of covering knowledge for a reasoning step. Indeed, the product-feature node N_x that contains the set of relevant products Π_x for a given set of features Φ_x (i.e. covering) can be very efficiently calculated: it is enough to find the most general largest equivalent node (*len*) whose node_features contains Φ_x , i.e.

$$N_x = len((\Phi_x, \Pi_x)). \quad (5)$$

2.4.2 Differentiating knowledge

The differentiate-task uses differentiating knowledge in order to eliminate some covering explanations generated in the cover-task. Obviously, the more explanations that are eliminated by using a differentiating knowledge, the more usability of that differentiating knowledge. In an

ideal case, after applying this knowledge only one explanation should remain. In the *c&d* method such knowledge is elicited from experts in a highly interactive process of refining the knowledge base [5].

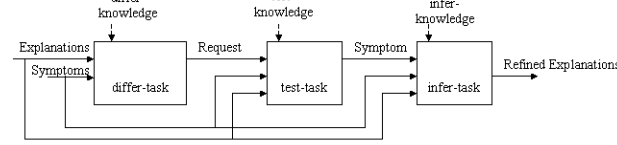


Figure 2. The decomposition of the differentiate task of *c&d* PSM

Analogy to the *c&d* method, the differentiate-task in the e-shopping scenario consists of three subtasks represented in Figure 2. The main problem is how to obtain the knowledge employed in these subtasks, which we discuss in next three subsections.

2.4.2.1 Differ knowledge

This subtask finds out which new symptoms (features) should be tested (e.g. is the value of the symptom X equal Y). In a problem-solving system this task can be seen as the crucial one: a system seems to be more intelligent if it makes as less as possible tests/questions in order to conclude something.

From the knowledge level point of view, the realization of this subtask should be driven by the “principle of rationality” of the agent – in the case of a shop agent this is to eliminate as much as possible irrelevant candidates. In other words, the selection of the features for testing should be done in such a way that the results of tests would enable maximal restriction of the searching space. It is clear that the selection of features for testing has to be *fair (complete)* – each candidate (relevant product) has a chance to “survive”. Moreover, it is clear that the number of tests should be *minimal*, since we can theoretically ask for the availability of each feature. In the e-shop domain the principle of the minimality is important due to a need to develop a buyer’s information need incrementally, in the so called step-by-step manner [13]. Briefly, on-line buyers often have a vague idea what to buy (due to the unfamiliarity with the content of the product database) or how to express their request (due to the unfamiliarity with the used vocabulary). In that case a shop assistant should ask only for features that do not imply another feature not yet considered by a user. A solution for this problem is to ask an expert which features should be tested in which situation. However, it can be a very expensive process and cannot guarantee a fair and minimal testing.

Another possibility is to reuse knowledge employed in the differ-task in the *c&d* method in e-shopping problem-solving. Basically, the *c&d* differ-knowledge compares *competing explanations* for a symptom directly, i.e. it compares each two explanations which cover the same set

of symptoms. Therefore, for a set of initial symptoms $c \& d$ differ-knowledge calculates the set of competing explanations which cover all symptoms and then tries to eliminate some of them by asking for their availability [14]. We use the same idea: for a set of product’s features Φ_{init} and the set of relevant products Π_{init} we calculate the set of possible competing features Φ_{com} , i.e. the set of features which can be found in relevant products. Using the notation introduced in the previous section, we can formalize this calculation as $\Phi_{com} = (\bigcup_{N_a <_{dir} N_l} \Phi_a) \setminus \Phi_{init}$,

(6)

where N_l is the Largest equivalent node (*len*) for the node $(\Phi_{init}, \Pi_{init})$. Therefore, the calculation is based on considering the features found in *direct_child* nodes of the Largest equivalent node of the node $(\Phi_{init}, \Pi_{init})$.

2.4.2.2 Test knowledge

The task of this knowledge is to perform tests on the features selected in the differ-task. In the simplest case a user is asked for the values of selected features. However, there are several methods that can be used to decide which products to eliminate. For example, several products can be compared, or the features of a product should be compared with each other. In all these tasks the test knowledge is used in order to perform tests in the most appropriate manner (from the user’s point of view).

Note that our approach is based on testing feature-value pairs. It means that we do not ask a user to choose between all values of a feature in the case that some of these values depend on the values of some other features. In that way we ensure minimal testing. Note that most of the other methods for generating product catalogues do not treat such a kind of dependencies between features.

2.4.2.3 Infer knowledge

This kind of knowledge enables the interpretation of test results in order to eliminate irrelevant results. Since the infer-task “understands” what is the goal of the whole method, it can interpret the test results differently depending on which strategy for elimination is selected.

3. Implementation & Evaluation

The research presented in this paper is a part of the Librarian Agent [15], a management system we have developed for the improvement of the search process in ontology-based information portals. The Librarian Agent is developed using the KAON ontology engineering framework. The e-ShopAgent is an extension of the Query Management module (dedicated to the query refinement) of the Librarian Agent. The visualisation metaphor is taken from the Librarian Agent.

Since the goal of our research is to model an efficient e-shopping support, our evaluation study concerns the comparison in the effectiveness (regarding searching) between a traditional e-shopping portal and a system based on the eShopAgent. Indeed, we compared searching for relevant products (cars) in two portals based on the same data, whereas one implements a traditional Interactive Query Refinement approach (IQE) [16] and another is based on the described approach. We selected 10000 cars from the actual offer from a car shopping portal (www.autocsout24.de) as instances in the repository. Each car had in average 15 features. In order to enable a fair comparison both approaches have been implemented using the same graphical interface, i.e. as KAON portals.

We compared four *formal* properties of a portal:

1. *Completeness of results in a step* – are all relevant products for a user’s query found by the system?
2. *Soundness of results in a step* – are only relevant products for a user’s query found by the system?
3. *Completeness of questions (query refinements) in a step* – are all relevant questions provided to the user in a refinement step?
4. *Minimality of questions* – are the provided question non-redundant?

We compared the navigation structure for 100 queries posted against both portals. The queries were selected by 10 participants (10 queries per a candidate) who actually have performed a search for a holiday trip. The participants were graduate students and no additional instructions were given to them. In order to ensure a fair comparison a half of tasks (searching), for each participant, was performed on each of portals.

Table 1 Results from the first evaluation study

Method	Completeness of results in a step	Soundness of results in a step	Completeness of questions in a step	Minimality of questions
IQE	85%	100%	50%	60%
Our	100%	100%	100%	100%

We made a post festum analysis of the support for the query refinement provided by a portal, by measuring parameters 1. - 4. for each navigation step in each navigation session. It means that we “traversed” off-line all navigation paths given by users and calculated (per hand) parameters 1. – 4. in each step. Table 1 summarizes the results. In order to simplify calculation (but without effecting the generality/validity of the experiment) we made a relative measurement, i.e. we put the parameters of a portal in the context of another. For example, for the parameter 3. we compared the set of questions provided by both portals. 100% means that this portal for that parameter includes all values produced by other portal.

Discussion: It is clear that in each refinement step a user can expect only relevant results (column 2: Soundness of results = 100%) even in a “traditional” portal. However, in the “traditional” portal some of relevant results are missing (column 1: about 15%), due to problems in modelling hierarchically organized data in a standard relational database. On the other side, in the semantic portal the transitivity axiom (from the ontology) ensures the completeness of the answers. Moreover 50% of relevant refinements that should be provided to a user (questions) are missing (column 3), what can be expected, since the refinement structure in a traditional portal is generated in an ad-hoc manner. Better results for Completeness of results can be explained by the fact that some products are placed in several refinements, so a user can find a product using several refinements. Finally, ad-hoc generation of refinements in traditional portals disables fine-tuning of user needs in a step-by-step manner in about 40% of cases (column 4), i.e. in 40% of refinements a user is provided with sub optimal recommendations for a refinement (e.g. a user is asked for a value of a product’s feature which can be derived from another features).

4. Related Work

Due to nature of the work, we tried to present the main differences between our approach and related product catalog approaches directly after introducing our ideas, so that the analysis of the related work is somehow distributed through the paper. We give here only a short analysis of the work related to query refinement since our approach can treated in that way as well.

In [17] the authors described an approach, named REFINER, to combine Boolean information retrieval and the content-based navigation with concept lattices. For a Boolean query REFINER builds and displays a portion of the concept lattice associated with the documents being searched centred around the user’s query. The cluster network displayed by the system shows the result of the query along with a set of minimal query refinements/enlargements. A similar approach is proposed in [18], by adding the size of the query result as an additional factor of the navigation. Moreover, the distance between queries in the lattice is used for similarity ranking. However, none of them put the concept lattice in a broader application context. Regarding searching in product catalogues the most similar approach is presented in [19]. It is an extension of a mediator architecture that supports the relaxation or tightening of query constraints when no or too many results are retrieved from the catalogue. The query language is a type of Boolean queries suitable for the (web) form based querying against

product catalogues. The query tightening is enabled when the cardinality of the resulted set has reached a predefined threshold and it is realized by selecting the most informative, not yet constrained product features. The information content of a feature is defined by measuring its entropy. However, this approach does not treat the problem of query refinement on the conceptual level, as our approach does. Finally, our approach can be seen as a method for Interactive Query Refinement for the case of logic-based information retrieval. In that sense our recommendations can be treated as a combination of subject thesauri and co-occurrence term lists [Sch96]. However, due to our scenario we extended existing methods for implicit relevance feedback.

5. Conclusion

In this paper we presented a general framework for modelling a buying process on the knowledge level, using the *cover and differentiate* problem solving method. We defined a reasoning pattern for an e-shop assistant, which models knowledge how to eliminate as much as possible irrelevant products using as less as possible questions in a refinement step. The role of a generic reasoning pattern in this scenario is not (only) to support reusability of a concrete solution, but to define the model in which all extensions of that reasoning pattern can be interpreted. In a case study we illustrated one of very important advantages of the proposed approach: the possibility to compare product catalog applications on the conceptual level.

Acknowledgement. Research for this paper was partially financed by BMBF in the project "SemIPort" (08C5939).

6. References

- [1] H. Shimazu "ExpertClerk: Navigating Shoppers Buying Process with the Combination of Asking and Proposing, IJCAI 2001, Morgan Kaufmann, San Francisco, pp.1443-1448, 2001.
- [2] M. Balabanovic, Y. Shoham: Content-Based, Collaborative Recommendation. CACM 40 (3), pp. 66-72, 1997.
- [3] A. Newell. The knowledge level. Artificial Intelligence, 18, pp. 87-127, 1982.
- [4] A.Th. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde: CommonKADS: A Comprehensive Methodology for KBS Development. In: IEEE Expert, December 1994, pp. 28-37.
- [5] S. Marcus, Automating Knowledge Acquisition for Expert Systems. Kluwer Academic Publishers, 1988.
- [6] N. Stojanovic, "Information-need Driven Query Refinement", The 2003 IEEE/WIC Conference on Web Intelligence (WI 2003), Halifax, Canada, IEEE Press, 2003.
- [7] B. J. Wielinga, A. Th. Schreiber, and J. A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1), Special issue 'The KADS approach to knowledge engineering', 1992.
- [8] B. Berendt, Using site semantics to analyze, visualize, and support navigation. *Data Mining and Knowledge Discovery*, 6, pp. 37-59, 2002.
- [9] W.K. Sung, D. Yang, S.M. Yiu, D.W. Cheung, W.S. Ho, T.W. Lam, S.D. Lee, Automatic Construction of Online Catalog Topologies, *IEEE Transactions on Systems, Man and Cybernetics – Part C*, V32, N4, pp. 382-391, Nov., 2002.
- [10] W.J. Clancey, The Knowledge Level Reinterpreted: Modeling How Systems Interact. In: *MachineLearning* 4, pp. 285-291, 1989.
- [11] N. Stojanovic, On Using Query Neighbourhood for Better Navigation Through a Product Catalog: SMART Approach, *IEEE International Conference on e-Technology, e-Commerce and e-Service*, Taiwan, IEEE Press, 2004
- [12] B. Ganter, R. Wille, Formal Concept Analysis: Mathematical Foundations, Springer Verlag, 1999.
- [13] P. Bruza, T. van der Weide, Stratified Hypermedia Structures for Information Disclosure. *The Computer Journal* 35(3): 208-220, 1992.
- [14] F. Puppe: Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods. Springer Verlag, Berlin, 1993.
- [15] N. Stojanovic, An Approach for Using Query Ambiguity for Query Refinement: The Librarian Agent Approach, 22nd International Conference on Conceptual Modeling (ER 2003), Chicago, Illinois, USA, Springer, 2003.
- [16] E.N. Efthimiadis, Interactive Query Expansion: a user-based evaluation in a relevance feedback environment. *Journal of the American Society for Information Science*, 51 (11), pp. 989-1003, 2000.
- [17] C. Carpineto, G. Romano, Effective re formulation of boolean queries with concept lattices. In *Flexible Query Answering Systems FQAS'98*, pp. 277-291, Berlin Heidelberg, Springer-Verlag, 1998.
- [18] P. Becker, P. Eklund, Prospects for Document Retrieval using Formal Concept Analysis, *Proceedings of the Sixth Australasian Document Computing Symposium*, Coffs Harbour, Australia, December 7, 2001.
- [19] F. Ricci, A. Venturini, D. Cavada, N. Mirzadeh, D. Blaas and M. Nones, Product Recommendation with Interactive Query Management and Twofold Similarity. 5th International Conference on Case-Based Reasoning (ICCBR 2003), 2003.
- [20] B. R. Schatz, E.H. Johnson, P.A. Cochrane, H. Chen, Interactive Term Suggestion for Users of Digital Libraries: Using Subject Thesauri and Co-occurrence Lists for Information Retrieval. *Digital Libraries*, pp. 126-133, 1996.

Predicting UML Statechart Diagrams Understandability Using Fuzzy Logic-Based Techniques

José A. Cruz-Lemus¹, Marcela Genero¹, José A. Olivas², Francisco P. Romero² and Mario Piattini¹

¹ALARCOS Research Group, ²ORETO Research Group

Department of Computer Science, University of Castilla-La Mancha

Paseo de la Universidad, 4, 13071, Ciudad Real (Spain)

{JoseAntonio.Cruz, Marcela.Genero, JoseAngel.Olivas, Mario.Piattini}@uclm.es

fpromero@soluziona.com

Abstract. In this work, we present an application of the Fuzzy Logic in the field of prediction in Software Engineering. We specifically use the Fuzzy Prototypical Knowledge Discovery for characterizing the UML statechart diagrams according to their understandability, starting from the structural complexity and size of the diagrams, expressed by means of metrics, and the Fuzzy Deformable Prototypes, to obtain a prediction model of the understandability time of the UML statechart diagrams. The obtained model, built from data obtained through experimentation, is valid –in a certain way- since the 75% of the estimated values are at least 70% accurate, although it is necessary further validation with data obtained from real projects.

1. Introduction

It is well-known in Software Engineering that the quality characteristics of object-oriented (OO) systems, such as maintainability, must be guaranteed from the initial stages of their lifecycle, focusing on the models obtained in these stages. In the recent years, this fact has been emphasized

given the great growth that the Model-Driven Development [1] and the Model-Driven Architecture [19] have experimented. In the OO development, some diagrams are done to cover static (class diagrams) and dynamic (use case diagrams, statechart diagrams...) aspects. For evaluating the quality of these diagrams in an objective way, it is necessary to rely on quantitative measures that avoid bias in the evaluation process.

There are several works published about quality measurement of UML class diagrams and use case diagrams [15]. However, there are only a few bibliographical references about metrics for behavioural diagrams, such as statechart diagrams, sequence diagrams or activity diagrams. Brito e Abreu et al. [9] and Poels and Dedene [24] pointed out that the definition of metrics for diagrams that capture dynamic aspects of OO systems is a relevant area for further research, but it has been disregarded in the software measurement field. This fact motivated us to define metrics for UML behavioural diagrams, starting with statechart diagrams [20] (see Table 1).

Table 1. Metrics for UML statechart diagrams

	Metric Name	Metric Definition
Size	NEntryA	The total number of entry actions, i.e., the actions performed each time a state is entered.
	NExitA	The total number of exit actions, i.e., the actions performed each time a state is left.
	NA	The total number of activities (do/activity) in the statechart diagram.
	NSS	The total number of states considering also the simple states within the composite states.
	NCS	The total number of composite states, i.e., the states with nested sub-states.
	NE	The total number of events.
Structural Complexity	NG	The total numbers of guard conditions.
	NT	The total number of transitions, considering common transitions (the source and the target states are different), the initial and final transitions, self-transitions (the source and the target states are the same) and internal transitions (transitions inside a state that respond to an event but without leaving the state).

	Metric Name	Metric Definition
	CC (McCabe's [18] Cyclomatic Complexity) ¹	Defined as $ NSS-NT +2$

Our approach about how the structural complexity and the size as internal attributes of statechart diagrams are potentially related with their understandability emerged from similar works [8][16] done in the field of Empirical Software Engineering, in which these properties were showed to be some of the greatest determinants of external quality characteristics, such as understandability and maintainability.

The metrics presented in Table 1 were theoretically validated using the Briand et al.'s [6] property-based framework, obtaining that the metrics NEntryA, NExitA, NA, NSS, NCS, NE and NG are size metrics, while NT and CC are complexity metrics.

As it is well-known in the software measurement field, if we want metrics that measure internal attributes (size, complexity...) to be useful, it is necessary that they can be used to predict some external attribute of quality, such as understandability or maintainability.

By means of a controlled experiment and its replication [15] that will be detailed in section 2, we have found out that the proposed metrics NA, NSS, NG and NT, seem to be strongly correlated with the understandability time the UML statechart diagrams. This led us to think about the construction of a prediction model of the understandability time of UML statechart diagrams, based on the values of these metrics. For the construction of the prediction model, we have used all the metrics, since we considered too premature to discard some of them.

Considering the encouraging results previously obtained by applying the Fuzzy-Prototypical Knowledge Discovery (FPKD) process and the Fuzzy Deformable Prototypes for the construction of prediction models applied to different domains [22][13][14], we decided to use them for our purpose. For the shake of brevity we will not explain in-depth all the steps of the prediction process, although further details about them can be found in [21].

In our case, the main goals of the prediction process are: firstly, the automatic search and extraction of fuzzy prototypes to characterize the UML statechart diagrams across their understandability, expressed as the structural complexity and size of the diagrams. This will be done using the Fuzzy Prototypical Knowledge Discovery

(FPKD) process; and secondly, the obtainance of a prediction model of the understandability time of the UML statechart diagrams, by means of the deformation of the previously discovered fuzzy prototypes.

The FPKD is an extension of the classic KDD [10] process that presents as novelties the incorporation of knowledge in different points by means of the user or the expert decisions and a result prepared to generate some conceptual prototypes called Fuzzy Deformable Prototypes, based on the idea of Fuzzy Prototypical Categories [21][27]. The use of fuzzy logic let us get these results in a more understandable and useful way for their later use in the prediction process. We can evaluate new situations from such prototypes, establish predictions for real situations and also make decisions from these predictions. Some other techniques, such as fuzzy clustering and aggregation functions [10], are also used, making easier the generation of structured, significant and easily updatable models.

This work is organized as follows: in section 2, the controlled experiment and its replication are presented. In section 3 we describe the different steps followed until getting the prediction model, which consist of the FPKD process, the proper prediction process and the validation of the prediction model. Finally, in section 4 we present the main conclusions and the future research lines emerged from this work.

2. Description of the data sources

In this section we will briefly describe a controlled experiment and its replication. They were carried out taking into account some suggestions provided by experts in Empirical Software Engineering [7][17][23][26]. Futher details of the exepriment and its repliaction can be found in [20].

2.1. First experiment

Using the GQM [2] template for goal definition, the goal of the experiment is detailed in Table 2.

¹Even tough the Cyclomatic Number of McCabe was defined to calculate single module complexity and entire system complexity, we tailored it for measuring the structural complexity of UML statechart diagrams.

Table 2. Goal of the experiment.

Analyze	<i>Structural complexity and size metrics for UML statechart diagrams</i>
For the purpose of	<i>Evaluating</i>
With respect to	<i>the capability of being used as indicators of the understandability of UML statechart diagrams</i>
From the point of view of	<i>researchers</i>
In the context of	<i>Undergraduate students of Computer Science and Software Engineering teachers of the Computer Science Department at the University of Castilla-La Mancha</i>

The experiment consisted of 20 UML statechart diagrams related to different universes of discourse but easy enough to be understood by each of the subjects (see an example in Appendix A). Each diagram had a test enclosed, which included a questionnaire in order to evaluate whether the subjects had really understood the content of the UML statechart diagrams. Each questionnaire contained four questions, each of these conceptually similar and written in the same order. Each subject had to write down the time he/she started and finished answering the questionnaire. The difference between these two values, expressed in seconds, is what we called ‘understandability time’. The subjects were given the material and they have to solve tests alone. We allowed them one week to return the experiment solved.

2.2. Experiment replication

The main differences between the experiment and its replication are:

- The subjects were twenty students enrolled in the third-year of Computer Science. Therefore, the subjects experience was lower than in the first experiment.
- They had to complete the tests alone and in no more than two hours. Any doubt could be solved by the person that coordinated the experiment, what contributed to control the plagiarism.

3. Building a prediction model for the understandability time of UML statechart diagrams

In order to build the prediction model, we carried out two main processes. First, a FPKD process, which consists of several steps: data transformation; obtainance of the prototypes using clustering techniques; parametric definition of the prototypes; fuzzy representation of the prototypes (using the data obtained in the first experiment). Then, we carried out a Prediction process, which consists of the ‘deformation’ of the fuzzy prototypes for predicting the understandability time of UML statechart diagrams and the Validation process (using the data of the replication).

Next, we will describe how we carried out each of these steps.

3.1. Data transformation

Firstly, it was necessary to transform the data so that they were valid for the FPKD process. On one hand, we obtained the table with the metric values for statechart. On the other hand, we obtained the understandability time for each diagram and subject. From these times, we obtained the minimum (MinUT), average (AvgUT) and maximum (MaxUT) understandability time of each diagram (see Table 3).

Table 3. Time obtained (in seconds) in the transformation process.

Diagram	AvgUT	MinUT	MaxUT
1	110.00	15	420
2	95.00	30	170
3	191.94	61	360
4	163.39	69	405
5	129.50	30	215
6	124.56	58	310
7	154.05	72	300
8	140.00	50	360
9	131.79	70	300
10	85.21	50	180

Diagram	AvgUT	MinUT	MaxUT
11	153.16	85	360
12	86.37	50	180
13	88.05	35	300
14	136.05	44	360
15	152.22	85	420
16	140.05	50	300
17	108.63	59	195
18	154.89	65	265
19	84.26	40	180
20	85.84	42	140

3.2. Obtainance of the prototypes using clustering techniques

With the aim of detecting the relationships between the UML statechart diagrams to be able later to ascertain whether they have a low, medium or high

understandability time, we will carry out a hierarchical clustering process, in the way of *Repertory Grids's* technique [3].

The diagrams were grouped in three prototypes according to the values of the metrics that reflect their structural complexity and size (see Table 4).

3.3. Parametric definition of the prototypes

Considering the data prototypes found in the previous section and their values of the understandability time

shown in Table 3, we obtained the parametric definition of the prototypes, as Table 5 shows.

3.4. Fuzzy representation of the prototypes

The three prototypes were represented as ‘fuzzy numbers’, which would allow us to obtain a degree of membership (between 0 and 1) of a new statechart diagram with each of the prototypes. To use triangular fuzzy numbers it is only necessary to know their centre and the size of the base of the triangle (named *Centre*, *a* and *b* in Table 6).

Table 4. Diagrams grouped in prototypes

Prototypes	Diagrams
Low Understandability Time	10,13,19
Medium Understandability Time	1,2,4,5,8,9,12,14,16
High Understandability Time	3,6,7,11,15,17,18,20

Table 5. Parametric definition of the prototypes

H: High Underst. Time		M: Medium Underst. Time		L: Low Underst. Time	
Average	2 min. 15 sec.	Average	2 min. 5 sec.	Average	1 min. 25 sec.
Maximum	7 min.	Maximum	7 min.	Maximum	6 min.
Minimum	42 sec.	Minimum	15 sec.	Minimum	35 sec.

Table 6. Fuzzy definition of the prototypes

Prototypes	Diagrams	a	Centre	B
Low Understandability Time	10,13,19	0	0.08	0.74
Medium Understandability Time	1,2,4,5,8,9,12,14,16	0	0.26	0.92
High Understandability Time	3,6,7,11,15,17,18,20	0	0.34	1

The formal definition of the prototypes as fuzzy numbers is obtained by means of a normalization process,

carried out in the following way $x'_n = \frac{x_n - x_{\min}}{x_{\max} - x_{\min}}$, and

the aggregation by means of average of the data corresponding to the metric values.

3.5. Deformation of the fuzzy prototypes to predict the understandability time of UML statechart diagrams

In this section we will show how to predict the understandability time for a new statechart diagram. We use as example the diagram 16 used in the experiment (showed in Appendix A)

The process is as follows:

1. Normalization of the values measured by means of the indexes of normalization associated with the obtained prediction model. The same formula is used as in the definition of the fuzzy numbers and with the same coefficients of minimum and maximum. In this way we obtained the values shown in Table 7.
2. Calculate the average of the previously normalized values (this value is called *X*). $X=0.53$.

3. From *X*, we obtain the degrees of membership to the prototypes represented by means of the fuzzy numbers as follows:

$$X > centre_{pi} \Rightarrow \mu_{pi} = \frac{X - a_{pi}}{centre_{pi} - a_{pi}}$$

$$X \geq centre_{pi} \Rightarrow \mu_{pi} = \frac{c_{pi} - X}{c_{pi} - centre_{pi}}$$

The results for the diagram 16 are shown in Table 8.

4. To obtain the predicted value of the understandability time for a new statechart diagram, the fuzzy prototypes are ‘deformed’ to consider the affinity degree with all the prototypes. Applying the concept of Fuzzy Deformable Prototypes defined in [21], the characterization of the proposed new statechart diagram can be described by the following linear combination:

$$C_{real}(w_1...w_n) = \left| \sum \mu_{pi}(v_1...v_n) \right|$$

Where:

C_{real} Real case proposed.

$(w_1...w_n)$ Parameters that describe the real case proposed.

μ_{pi} Degree of membership with the non-zero Fuzzy Deformable Prototypes.

$(v_1 \dots v_n)$ Parameters of these Fuzzy Deformable Prototypes.

For the diagram 16 the predicted value is shown in Table 9.

The result of applying the prototype deformation to every diagram is shown in Table 10.

Table 7. Metric values for the diagram 16.

	NEntryA	NExitA	NA	NSS	NCS	NT	NE	NG	CC
Values	0	0	5	9	0	21	22	1	16
Normalized	0	0	1	0.7	0	1	0.95	0.3	1

Table 8. Value of the affinities of the diagram 16 with the prototypes.

Prototypes	Affinities
Low Understandability Time	0
Medium Understandability Time	0.591
High Understandability Time	0.712

Table 9. Predicted value for the diagram 16².

Average		2 min. 5 sec.			2 min. 15 sec.		2 min. 2 sec.
Maximum	0.591 / 2	7 min.	+	0.712	7 min.	=	7 min. 5 sec.
Minimum		15 sec.			42 sec.		34 sec.

Table 10. Predicted values for each UML statechart diagrams.

DIAGRAM	X	Aff(B)	Aff (M)	Aff (A)	Estimated value	Real value	MRE
1	0.15	0.894	0.577	0.441	116.38	110.00	0.058
2	0.14	0.909	0.538	0.412	114.925	95.00	0.210
3	0.18	0.848	0.692	0.529	120.52	191.94	0.372
4	0.16	0.879	0.615	0.471	117.765	163.39	0.279
5	0.24	0.758	0.923	0.706	162.75	129.50	0.257
6	0.41	0.5	0.773	0.894	156.41	124.56	0.256
7	0.23	0.773	0.885	0.676	158.9375	154.05	0.032
8	0.34	0.606	0.879	1	177.875	140.00	0.271
9	0.23	0.773	0.885	0.676	158.9375	131.79	0.206
10	0.11	0.955	0.423	0.324	110.785	85.21	0.300
11	0.34	0.606	0.879	1	177.875	153.16	0.161
12	0.12	0.939	0.462	0.353	112.155	86.37	0.299
13	0.06	0.75	0.231	0.176	79.92	88.05	0.092
14	0.1	0.97	0.385	0.294	109.4	136.05	0.196
15	0.39	0.53	0.803	0.924	162.485	152.22	0.067
16	0.53	0.318	0.591	0.712	122.205	140.05	0.127
17	0.18	0.848	0.692	0.529	120.52	108.63	0.109
18	0.51	0.348	0.621	0.742	125.555	154.89	0.189
19	0.05	0.625	0.192	0.147	66.565	84.26	0.210
20	0.16	0.879	0.615	0.471	117.765	85.84	0.372

3.6. Validation of the prediction model

We based on the most commonly used techniques [11] to evaluate the accuracy of our prediction model, MMRE, MdMRE and Pred(25%).

The values of MRE obtained for each diagram are shown in Table 10. In this experiment, the value for

MMRE and MdMRE for is 0.20. The value obtained for Pred(25%) is a 70%, what indicates that a 75% of the obtained values are at least 70% accurate.

In Figure 1 are shown the predicted and real average values for the understandability time of each statechart diagram of the experiment.

² In this case, following some recommendations given by the experts, as the sum of the membership degrees is greater than 1, we divided the membership degree of the second most similar prototype by two.

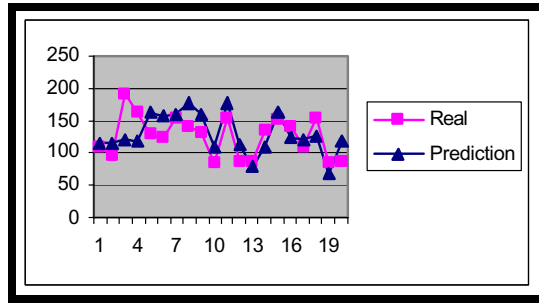


Figure 1. Predicted values vs. real values

4. Conclusions

The main contribution of this work is a prediction model for the Understandability Time of UML statechart diagrams. This model was built from some metrics for the structural complexity and size of UML statechart diagrams using two fuzzy logic-based techniques: the Fuzzy Prototypical Knowledge Discovery (FKPD) process and the Fuzzy Deformable Prototypes. The data used to build the model was obtained through a controlled experiment. Moreover, the model was validated using data obtained in a replication of the experiment. Through the validation, we reached the conclusion that – in a certain way – it is a good model, since a 75% of the understandability time estimated values are at least 70% accurate.

Although the results are encouraging, we are aware that we must improve our study in two ways: with respect to the data used for obtaining the prediction model and with respect to the technique applied for building the prediction model.

For that, on one hand we have to replicate the experiment with professionals and examine the usefulness of the metrics in real projects. Related to the prediction model, there are also some aspects to improve. Using algorithms such as Fuzzy C-Means [4], Fuzzy Kohonen Networks [5] or soft clustering algorithms in general, would allow us to raise the power of problems resolution. These algorithms can make the clustering process and the model construction to be done at once, deciding the number of prototypes before being carried out. Moreover, these algorithms allow a better manipulation of great volume of data.

Acknowledgements

This research is part of the MESSENGER project (PCC-03-003-1) financed by “Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha (Spain)” and the CALIPO project supported by “Dirección General de Investigación del Ministerio de Ciencia y Tecnología (Spain)” (TIC2003-07804-C05-03).

References

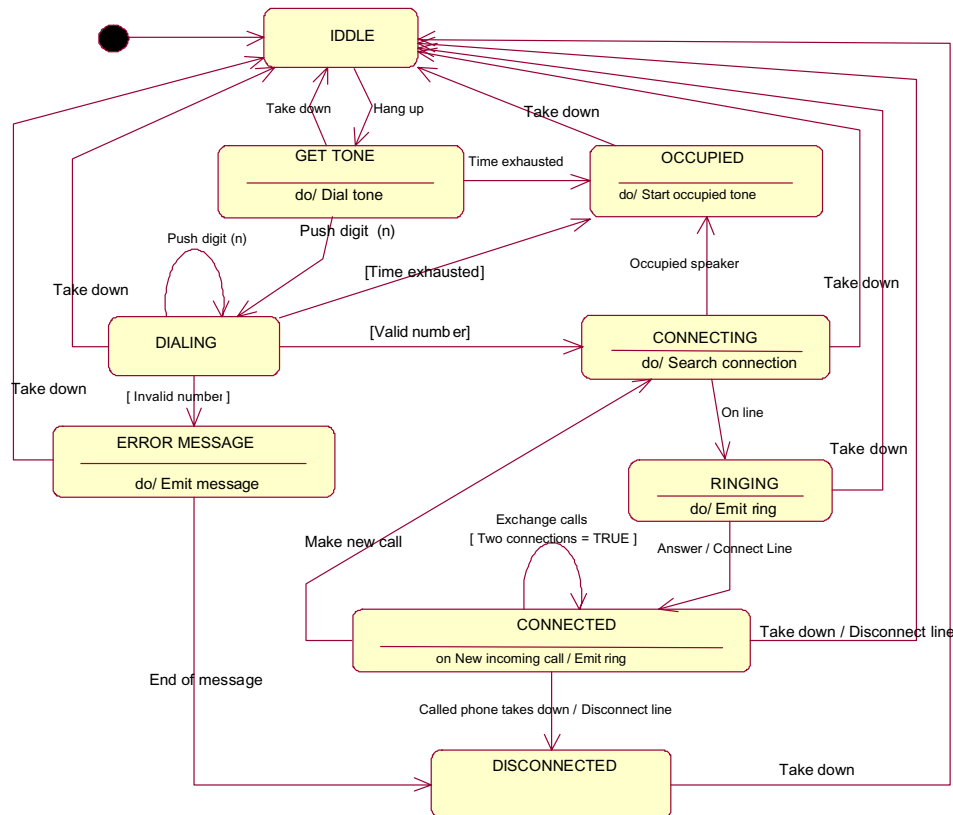
- [1] Atkinson C. and Kühne T. (2003). “Model-Driven Development: A Metamodeling Foundation”. *IEEE Software* 20(5), 36- 41.
- [2] Basili, V. R., Caldiera, G. y Rombach, H. D. (1994). *Goal Question Metric Paradigm*. Encyclopedia of Software Engineering, vol. 1. John Wiley & Sons, 528-532.
- [3] Bell R. (1990). “Analytic Issues in the Use of Repertory Grid Technique”. *Advances in Personal Construct Psychology* 1, pp. 25-48.
- [4] Bezdek J., Hathaway R., Sabin M., Tucker W. (1987). “Convergence Theory for Fuzzy c-Means Counterexamples and Repairs”. *IEEE Trans Syst., Man and Cybern.* SMC-17 (5), pp. 873 - 877.
- [5] Bezdek J., Tsao E., Pal N. (1992). “Fuzzy Kohonen Clustering Net-works”. *IEEE International Conference on Fuzzy Systems*. San Diego, pp. 1035-1043.
- [6] Briand, L., Morasca, S., Basili, V. (1996) “Property-based software engineering measurement”. *IEEE Transactions on Software Engineering*, 22 (1) pp. 68-85
- [7] Briand L., Arisholm S., Counsell F., Houdek F., Thévenod-Fosse P. (1999b). “Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions”. *Empirical Software Engineering*, 4(4), pp. 387-404.
- [8] Briand L., Bunse C., Daly J. (2001). “A Controlled Experiment for evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs”. *IEEE Transactions on Software Engineering*, 27(6), pp. 513-530.
- [9] Brito e Abreu F., Zuse H., Sahraoui H., Melo W. (1999). “Quantitative Approaches in Object-Oriented Software Engineering”. *ECOOP’99 Workshops*, LNCS 1743, A. Moreira and S. Demeyer (eds). Springer-Verlag, pp. 326-337.
- [10] Castro J., Trillas E., Zurita J. (1998). “Non-monotonic Fuzzy Reasoning”. *Fuzzy Sets and Systems* 94, North Holland, pp. 217 - 225.
- [11] Conte S., Dunsmore H., Shen V. (1986). *Software Engineering Metrics*. Benjamin-Cummings Publishing Co., Inc., USA.
- [12] Fayyad U., Piatetsky-Shapiro G., Smyth P. (1996). “The KDD Process for Extracting Useful Knowledge from Volumes of Data”. *Communications of the ACM*, 39(11), pp. 27 - 34.
- [13] Genero M., Olivas J., Piattini M., Romero F. (2001). “Using metrics to predict OO information systems maintainability”, *CAISE 2001*, Lecture Notes in Computer Science, 2068, Interlaken, Switzerland, 388-401.
- [14] Genero M., Piattini M., Calero C. (2002). “An study to validate metrics for class diagrams”. *Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes de Software (IDEAS’2002)*, La Habana (Cuba), pp. 226-235.

- [15] Genero M., Piattini M. and Calero M. (Eds.) *Metrics For Software Conceptual Models*. Imperial College Press, UK, 2004.
- [16] Harrison R., Counsell S., Nithi R. (2000). "Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems", *The Journal of Systems and Software*, 52, 173-179.
- [17] Kitchenham B., Pflegger S., Pickard L., Jones P., Hoaglin D., El-Emam K. y Rosenberg J. (2002). "Preliminary Guidelines for Empirical Research in Software Engineering". *IEEE Transactions of Software Engineering* 28(8), pp. 721-734.
- [18] McCabe, T. (1976). "A Complexity Measure". *IEEE Transactions on Software Engineering*. Vol. 2. N°4, pp. 308-320.
- [19] MDA- The OMG Model Driven Architecture (2002). Available: <http://www.omg.org/mda/>, August 1st, 2002.
- [20] Miranda D., Genero M., Piattini M. (2003). "Empirical validation of metrics for UML statechart diagrams". *Fifth International Conference on Enterprise Information Systems (ICEIS 03)*, 1, pp. 87-95.
- [21] Olivas J. (2000). *Contribución al Estudio Experimental de la Predicción basada en Categorías Deformables Borrosas*, Tesis Doctoral, Universidad de Castilla La Mancha, España.
- [22] Olivas J., Romero F. (2000). "FPKD. Fuzzy Prototypical Knowledge Discovery. Application to Forest Fire Prediction". *Proceedings of the SEKE'2000*, Knowledge Systems Institute, Chicago, Ill. USA, pp. 47 - 54.
- [23] Perry D., Porter A., Votta L. (2000). "Empirical Studies of Software Engineering: A Roadmap". *Future of Software Engineering*. Ed:Anthony Finkelstein, ACM, pp. 345-355.
- [24] Poels, G. and Dedene, G. (2000). "Measures for Assessing Dynamic Complexity Aspects of Object-Oriented Conceptual Schemes". *Proceedings of 19th International Conference on Conceptual Modelling (ER 2000)*, pp. 499-512.
- [25] Schneidewind N. (2002). "Body of Knowledge for Software Quality Measurement". *IEEE Computer* 35(2), pp. 77-83.
- [26] Wohlin C., Runeson P., Höst M., Ohlson M., Regnell B., Wesslén A. (2000). *Experimentation in Software Engineering: An Introduction*, Kluwer Academic Publishers.
- [27] Zadeh, L. A. (1982). "A note on prototype set theory and fuzzy sets". *Cognition* 12, pp. 291- 297.

In this appendix we will show, as example, one of the test used in the experiment, corresponding to the diagram 16.

Appendix A

DIAGRAM 16: MAKING A TELEPHONE CALL 2



TIME NOW: _____

Answer the following questions:

1. If you get from CONNECTED to IDDLE, which event has occurred previously? **TAKE DOWN**
2. If you are CONNECTED and the event *Occupied speaker* occurs, which state do you get to?
OCCUPIED
3. Which events and/or conditions will have occurred at least and in which order for getting from IDDLE to RINGING?
(1) Hang up (2) Push digit(n) (3) [Valid number] (4) On line
4. Starting from ERROR MESSAGE, which state will you get if the following sequence of events and conditions occurs? (1) Take down (2) Hang up (3) Push digit(n) and (4) [Time exhausted]. **OCCUPIED**

TIME NOW: _____

Programming ubiquitous software applications: requirements for distributed user interface

Anders Larsson and Erik Berglund
Department of Computer and Information Science
Linköping University
Linköping, Sweden
andla@ida.liu.se, eribe@ida.liu.se

Abstract

Mobile and ubiquitous computing require new approaches to user interface design. Incorporating I/O devices in the environment is imperative because small devices do not provide enough interaction richness. Distributed user interfaces (DUIs) are needed to take advantage of such I/O-landscapes.

A DUI constitutes a fundamental change of the pretext of user interface development. New programming models that support efficient creation and maintenance may be required. This paper presents a case study in DUI design and report on the use of current GUI modeling techniques to provide DUIs. We identified several issues where current programming models need to be extended.

1. Introduction

Mobile, ubiquitous computing is an area of growing importance, where computing power is transferred from static locations to mobile or ubiquitous platforms [28, 8, 10, 20]. One important aspect of interaction in these circumstances is context-aware computing where the situation has bearing on the user interface (UI) of an application and where situated action [23] thereby becomes integrated in UIs. We are currently seeing an increased focus on mobile and ubiquitous solutions to traditionally desktop applications [14, 16, 27].

This focus mobile, ubiquitous applications place new requirements on UI development. Limited interaction richness is provided by smaller computers, hand held or integrated in everyday artifacts. As a consequence, a limited set of functionality can be supported using traditional UI development methods. New UI programming models are required to express functionality. Work has, for instance, been performed specifically on the design of graphical user

interfaces (GUIs) for small devices [26].

An alternative approach to traditional GUIs interface design is *distributed user-interfaces* (DUI) [4], where mobile devices are augmented with external devices located in the work environment or even where mobility is created solely by such an I/O landscape (a geography of I/O devices that may change over time). A complexity in mobile and ubiquitous computing lies in the fact that the physical resources users work through change over time. Abowd calls this the interface scalability problem [1]. DUIs solves this problem by describing how the UI and the functionality it represents should change for variable situations.

In essence, DUIs constitute dynamically configured peer-groups of UI components found in I/O landscapes that multiple concurrent users access simultaneously. Using DUIs, application can take advantage of, but must also adapt to, variable interaction functionality being present during execution. Sometimes there is an abundance of interaction richness and at other times only limited interaction is available. This lead to new degrees of both freedom and complexity in the design of application UIs.

Developers need new programming models to adequately handle DUI construction. Without adequate programming models and tools support for UI development, software development for mobile and ubiquitous computing is both difficult and time consuming [18]. While DUI construction is a feasible technical concept, it is not equally clear how a DUI programming model and suite of supporting tools should be designed.

This paper discusses programming models for DUI construction with mobile and ubiquitous computing in mind. The work is based on the LINDA-2 case study, where a healthcare groupware system is designed for a number of different I/O configurations. Based on our experiences, we provide a general discussion of the extension required to current GUI programming models (found in programming languages such as Java and Visual C++) needed to ade-

quately support DUI development.

The paper is organized in the following sections :

- Background (User Interface Programming Model, Distributed User-interfaces)
- Method (Developing DUIs: Case Study, Related Work)
- Result (Requirements on DUIs Models, Conclusion)

2. User Interface Programming Models

Programming models and tools that implement these models are required to create and maintain UIs. Large portions of the software development cost and time are spent constructing the user interface. Myers reported in 1994 that almost all systems developed for Unix involved a graphical user-interface (GUI) and that the GUI part of the source files consumed about half of the total code. The time spent building the GUI consumed the same amount of time as all the other code parts put together.

The construction of GUIs can be considered very difficult and is usually performed iteratively to get reliable results. Compared to command line UIs, GUIs force programmers to work with a more complex interaction structure where elaborate graphics, multiple interaction mechanisms, and non-linear command structures. [18] We may have to expect the same increase in complexity moving from GUIs to DUIs.

The fact that UI construction is problematic and time consuming has resulted in the development of programming models to support this task, such as MFC (Microsoft Foundation-Classes), Java Swing and Motif [4]. These models share a set of common instruments that contain:

- widgets (such as buttons, slide-bars, menus, and text-areas)
- interactive actions on widgets (general events related either to logical changes in program state or specifically to different I/O devices). Mainly focused on mouse and keyboard events as a consequence of the desktop metaphor.
- messaging structures among widgets or other software components (such as events).

Development of GUIs is also often performed using some sort of graphical construction tool, usually integrated in a development environment, such as Borland JBuilder, IBM VisualAge or Microsoft Visual C++. In empirical studies, tools have been shown to reduce the time spend creating GUIs by a factor of four, or reduced the number of lines of code by 83 % [18].

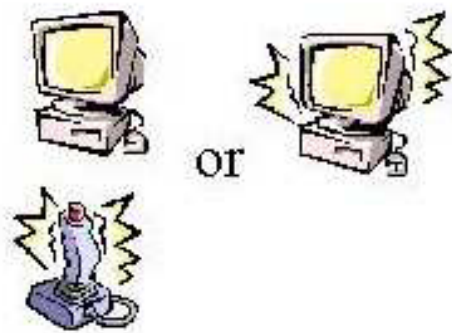


Figure 1. DUIs handle multiple I/O configurations.

3. Distributed User-interfaces

Distributed user-interfaces (DUIs) are application UIs that handle variable sets of interaction devices. A simple example of a DUI design is a movie player that automatically presents subtitles when speakers are missing. Another example is a video game that when a rumble-pack is missing from the hand-control shakes the screen image instead, see Figure 1. A more general description of a DUI is a UI where different I/O devices negotiate responsibility for interaction and where systems are prepared for change in interaction richness and the lack thereof.

DUIs are particularly relevant to mobile and ubiquitous computing where interaction devices found in the work environment are dynamically grouped to provide users with sufficient interaction richness and thereby enhance the ability of small devices. The following example scenario illustrates a more complex DUI mobile applications:

Lisa Eriksson is visiting a trade fair. She walks through the exhibition floor passing different booths. Simultaneously, she is surfing an exhibition web site with her mobile phone. As she approaches a booth in the exhibition area, her mobile phone presents the available multimedia and information services for that particular booth. The mobile phone provides a digital video clip, which she activates, and it starts playing on the mobile-phone display. However, the presence of a 50 inch screen is also detected and gets plugged into Lisas mobile web browser across the network. The available interface components compare their capabilities with regards to screen resolution and size and renegotiates responsibilities. The decision is reached to redistribute the digital video clip to the 50 inch screen while presenting

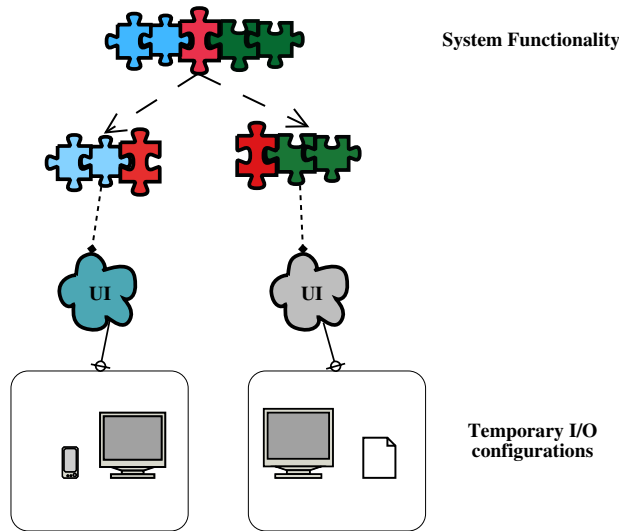


Figure 2. DUIs are based on temporary I/O configurations that provide interaction richness for UIs that in turn provide adequate representation of functionality.

a more advanced remote control GUI on the mobile phone display. As Lisa walks away, the video clip is removed from the 50 inch screen and starts playing in her mobile phone again. She stops the clip and leaves the booth.

For scenarios of this type, users are likely to actively locate adequate interaction devices when needed rather than carry them around. This makes room for mobility in computing even for interaction rich applications.

Essentially, DUI programming requires UIs that are functions of variable I/O configurations found in the I/O landscape. A first step is, however, to model concurrent UI versions in the development of DUI applications. Programmers must be able to define how applications change their interaction modalities and adapt their functionality with regards to different settings. Figure 2 illustrates DUI programming model basis.

Applications supporting DUI should be able to [4]:

- be split over a set of I/O-devices.
- adapt the functionality to fit the present devices.
- share devices with other applications.

A programming model for DUI is a sound way of constructing a UI that is a function of a variable set of I/O-devices. A first approach toward such a function is a tool-based means for handling a series of defined sets of I/O-

configuration. A DUI programming model allows developers to build such series efficiently with minimal code and with a good graphical representation that ensures that the UIs are correct in each configuration. The model should allow developers to easily create and maintain concurrent UI versions for variable sets of I/O-devices. This can be achieved by developing design patterns and development tools for user interfaces development.

4. Developing DUIs: Case Study

The conclusions of this paper are based on a DUI implementation case study using current GUI models. Mobility is in our setting based on I/O devices found in the environment. Here we present the case study.

4.1. LINDA2-case

The LINDA-2 [6] system is a DUI extension to an existing research group-ware system for the for a Swedish emergency medical work environment, the LINDA system [7]. In the medical environment, demands on system transparency are extremely high. Users need to work with minimal manual configuration from many different locations. In many ways the medical work environment is a highly relevant case environment for mobile, ubiquitous applications [4].

The current version of the LINDA-2 handles the following I/O configurations:

- Normal Workstation (Keyboard, Mouse and Screen)
- Ericsson ChatPen with headset
- Ericsson ChatPen with walk-up display
- Ericsson ChatPen with headset and walk-up display
- Ericsson ChatPen alone

The Ericsson ChatPen is a pen build with Anoto hardware that makes it possible to capture the pen-strokes in a digital form and send them over a network to a computer [2]. The problem of a paper-based interface to a computer system is feedback and inconsistency between paper and virtual document. In order to provide feedback to the user we needed to provided the users with a alternative ways if receiving system response. This was made possible by adding different feedback channels to the paper-clients using audio cues and/or visual cues.

The different UI parts of the system were using a architectural version of the MVC-design pattern (Model-View-Controller) [11], which allowed for an easy-way of sharing code-models between the different clients. Sharing code between the different interfaces, pen and paper together with

a feedback channel, made it possible to build components that could be used in a set of different DUI versions.

4.2. Current state and future Work for the LINDA-systems

The LINDA systems, the original LINDA-system and the ubiquitous extension LINDA-2, are currently used as a test-bed for research in CSCW (Computer-Supported-Collaborative-Work), DUI and ubiquitous computing. The future and ongoing work on the systems include both work on creating a more robust hybrid model between physical and virtual objects as well as a better editor that can be used for creating DUI interfaces. Parallel to this the research on a programming model for distributed interface will continue.

5. Related work

The research in DUI's is an area cross cutting several different and broad areas such as software engineering to ubiquitous computing. In ubiquitous computing a lot of work has been done for task-oriented systems at the Aura project at Carnegie Melon University. A task-oriented system is a distributed and mobile system to help users solve their current work-task. A user that is editing a document at work, the user's task is then editing, can bring his task with him as he goes home and can continue his task on the bus on his hand held device and as he gets home his home workstation can take over. The user never has to be concerned with moving files or starting up software, the system knows what the user task is and uses the best suited software available on the users current platform to aid the user in solving the task.[12, 21] Another project, also at Carnegie Melon University, is the Pebbles project [25], where handhelds and PC are used together. The projects aims to spread computing functions with their related user interfaces over different I/O-devices.[19]. Adaptive application, mobile application, multiple-device user interface (MUIs) and collaborative work applications are other areas of important for continued work of distributed user-interface. A lot of the work done in the mobile-, mui- and adaptive application area focus on making traditional desktop application available on small mobile devices such as cell phones or PDA's. Research is conducted in the area of programming API's for development of mobile applications, the construction of display logic languages, such as html, css, wml, xml and xslt, and as well as programming models and tools [17, 13]. As a part of the DUI is using the available devices located in the environment users may need to share certain devices, such as screens, research has shown several ways in which devices can be made sharable among users. This work is specially focused on several user trying to solve a common task using a set of devices, such devices configuration can

be two mice with one screen or one large screen and several laptops connected together. [3, 15, 22] Although these areas are not directly connected to one another they are all important for the future development of distributed user-interfaces.

6. Requirements on DUIs Models

From our work implementing DUIs we draw conclusions with regards to the requirements on DUI programming models. The baseline in this discussion is the GUI structures of common programming language such as Java and MFC [4]. These requirements are focused on a scenario where tool-supported developers graphically build DUIs as sets of concurrent GUIs. This is a first and perhaps also most likely DUI development model considering how GUI are developed today (i.e., through tool support). A future formal approach to DUI construction could perhaps abstracts away the manual labor of DUI construction.

In short we find the following issues worth discussing: interaction openness, disappearing I/O, widgets, events, and layout managers

6.1. Interaction Openness

For a UI where the setting for interactivity is relative, openness in programming models becomes an imperative aspect. Openness is the programming models ability to incorporate new components or new versions of existing components. Designing for every possible configuration is unrealistic and the programming model therefore need to be prepared for change.[24] The DUI programming model needs to be designed for interaction openness, allowing the UI to support different interaction styles, such as speech and gesture recognition, for the same logical interaction. Saying "ok" can then be handled as identical to clicking a "ok" button. Such interaction openness can be accomplished by introducing a device independent interaction structure which different component then implement.

6.2. Disappearing I/O

The move from command-line to traditional GUIs resulted in a more complex development task for programmers. The move from GUIs to DUIs will probably result in the same type of complexity increase. Programmers will not only have to consider a non-linear command structures but will also have to consider the fact that devices and interface might disappear as users walks around. Changing I/O settings require that degradation become a central part of programming.

Disappearing I/O is handled by exception handling in current programming models, i.e., treated as something un-

usual that breaks execution. In a DUI programming model, such degradation must be expected. DUIs should be designed to reflect contextual changes and not only survive system crashes or unprepared lack of resources. To a certain degree it is relevant to speak of a requirement for functional hibernation, that is functions must be removed safely when adequate interaction devices are not present. In this sense, method call structures may have to include parameters that identify the available I/O capability. An I/O management system could provide the status of the available resources.

6.3. Widgets

Most of the widgets found in GUI programming models of today provide sufficient graphical expressiveness for DUIs. However, they are not flexible enough for the new conditions. In today's GUI-toolkits there are sets of reusable components like buttons, text-areas, scrollbars. Few of these components support disappearing multiple input/output forms. Studies have, for example, show that augmenting button used on a PDA with sound when pressed (instead of a graphical animation) increased the speed with which users can work with the device [5]. Furthermore, widgets cannot but may have to be able to split themselves across several I/O devices in DUIs. In fact, application may require that certain information only be show on certain devices (e.g., for integrity reasons in a public space). Widgets need to be extended with multiple forms for different contexts and also provide the same type of openness as DUIs in general.

6.4. Events

The messaging structure among components in GUIs are events, issued by the system when interaction events occur (e.g., when a user clicks on a button). Currently, event structures are being designed for distributed components, components that reside on different computers. When the graphical user-interface no longer exists on a single address space, or even the same machine, the event structures found in today's models and languages need to be updated to support events that are distributed over a set of devices. Eugster, Baehni and Damm showed one way of doing this by so called obvents. Obvents are created as an extension to Java, to be used in distributed environments [9].

However, for DUIs extension of the current event structures beyond distributed systems event structures will be needed as well. DUIs event structures must not only associate actions with distributed components but also handle the fact that a widget itself may be located on multiple platforms. Events may also have to be context aware for an environment where several users and applications work concurrently.

6.5. Layout Managers

Layout managers are designed to handle the typesetting of components in GUIs; automating typographical rule sets to automatically provide a suitable organization of a GUI. For DUIs many of the premises of layout managers change, in particular when components are spread over multiple I/O devices. New means of relative positioning appear and developers must be able to specify how components are distributed, for instance, to make sure that unwanted distribution is prohibited. For instance, a button may have to be presented together with the choice box it is related to.

Layout managers can incorporate decision rules for the distribution of components and potentially be a placement for formal advances in DUI construction. Layout managers may also be resources for graphical tools but not be used directly by users. Judging from work on layout managers, it is a complex issue that requires much work. Finding a simple layout manager rule set for DUIs is a difficult task.

7. Conclusion

Distributed user-interfaces allow software developers to prepare for changing I/O configurations in interaction landscapes. Developers of mobile applications can modify looks and behavior according to the devices setting a user is currently using. This provides the necessary means for developing mobile applications that provide a high degree of interaction richness using resources in the environment. In order to build and maintain these applications in a sound and effective it is imperative that our programming model are designed to handle this; UI construction are among the most resource-expensive programming activities.

In the process of developing a paper interface for a existing health care system we uncovered programming model issues needed to create sound development processes and tools for DUIs:

- *Openness*, allowing for the addition of new configurations of devices.
- *Disappearing I/O*, providing messaging of the disappearance of I/O devices.
- *Widget* must be flexible with regards to appearance and directly support graceful degradation
- *Events* must handle widgets that them selves can be spread over many devices.
- *Layout managers*, formalizing the distribution of components across I/O components and allowing users to describe requirements of such a distribution.

A effective way of construction and maintain user interfaces in the past have been good tools. This will also be the case for distributed user-interfaces, perhaps even more important since the complexity increases.

References

- [1] G. Abowd. Software engineering and programming language considerations for ubiquitous computing. *ACM Computing Surveys (CSUR)*, 28(4es):190, 1996.
- [2] Anoto Group. *Anoto*. World Wide Web, <http://www.anoto.com>, 2002.
- [3] R. Bentley, T. Rodden, P. Sawyer, and I. Sommerville. An architecture for tailoring cooperative multi-user displays. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'92)*, pages 187–194, Toronto, Ontario, 1992. ACM Press.
- [4] E. Berglund and M. Bång. Requirements for distributed user-interface in ubiquitous computing networks. In *MUM2002, Mobile and Ubiquitous MultiMedia conference*, 2002.
- [5] S. Brewster. Overcoming the lack of screen space on mobile computers. *Personal and Ubiquitous Computing*, 6(3):188–205, 2002.
- [6] M. Bång, E. Berglund, and A. Larsson. A paper-based ubiquitous computing healthcare environment. In *Fourth International Conference on Ubiquitous Computing (UbiComp 2002)*, 2002.
- [7] M. Bång, A. Hagdahl, H. Eriksson, and T. Timpka. Groupware for case management and inter-organizational collaboration: The virtual rehabilitation team. In *World Congress on Medical Informatics (Medinfo 2001)*, 2001.
- [8] A. J. Demers. Research issues in ubiquitous computing. In *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*, pages 2–8. ACM Press, 1994.
- [9] P. T. Eugster, R. Guerraoui, and C. H. Damm. On objects and events. In *Proceedings of the OOPSLA '01 conference on Object Oriented Programming Systems Languages and Applications*, pages 254–269. ACM Press, 2001.
- [10] G. Forman and J. Zahorjan. The challenges of mobile computing. *Computer*, pages 38–47, April 1994.
- [11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1994.
- [12] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project aura: Towards distraction-free pervasive computing. *IEEE Pervasive Computing, special issue on "Integrated Pervasive Computing Environments"*, pages 22–31, April-June 2002.
- [13] J. Grundy and B. Yang. An environment for developing adaptive, multi-device user interfaces. In *Proceedings of the Fourth Australian user interface conference on User interfaces 2003*, pages 47–56. Australian Computer Society, Inc., 2003.
- [14] H. Ishii and B. Ullmer. Tangible bits: Towards seamless interfaces between people, bits and atoms. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, 1997.
- [15] I. Marsic. An architecture for heterogeneous groupware applications. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 475–484. IEEE Computer Society Press, 2001.
- [16] D. R. McGee and P. R. Cohen. Creating tangible interfaces by augmenting physical objects with multimodal language. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 113–119. ACM Press, 2001.
- [17] M. McIlhagga, A. Light, and I. Wakeman. Towards a design methodology for adaptive applications. In *Mobile Computing and Networking*, pages 133–144, 1998.
- [18] B. A. Myers. User interface software tools. *ACM Transactions on Computer-Human Interaction*, 2(1):64–103, 1995.
- [19] B. A. Myers. Using handhelds and pcs together. *Communications of the ACM*, 44(11):34–41, 2001.
- [20] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, pages 10–17, Aug. 2001.
- [21] J. Sousa and D. Garlan. Aura: an architectural framework for user mobility in ubiquitous computing environments. In *IEEE/IFIP Conference on Software Architecture*, 2002.
- [22] J. Stewart, B. B. Bederson, and A. Druin. Single display groupware: A model for co-present collaboration. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 286–293, 1999.
- [23] L. A. Suchman. *Plans and situated actions: the problem of human-machine communication*. Cambridge University Press, 1987.
- [24] P. Tandler. Software infrastructure for ubiquitous computing environments: Supporting synchronous collaboration with heterogeneous devices. In *Proceedings of UbiComp 2001: Ubiquitous Computing*, number 2201 in LNCS, pages 96–115. Springer Verlag, Heidelberg, 2001.
- [25] The Pebbles Project. The pittsburg pda project. <http://www-2.cs.cmu.edu/pebbles/>.
- [26] A. Uotila. A user interface toolkit for a small screen device. Master's thesis, University of Tampere, Department of Computer and Information Science, February 2000.
- [27] R. Want, K. P. Fishkin, A. Gujar, and B. L. Harrison. Bridging physical and virtual worlds with electronic tags. In *CHI*, pages 370–377, 1999.
- [28] M. Weiser. The computer for the 21st century. *Scientific American*, pages 94–104, September 1991.

Requirements Scenarios Based System-Testing

Ridha Khedri

Department of Computing and Software

McMaster University

1280 Main Street West, Hamilton, Ontario, Canada

khedri@mcmaster.ca

Imen Bourguiba

École Nationale des Sciences de l'Informatique

Université de la Manouba

Manouba, Tunisia

bourguiba@cas.mcmaster.ca

Abstract

System and acceptance testing, despite their importance, are probably the least understood specification-based testing phases. Testing from specifications offers testers and users confidence in the intended functionality of the system.

In this paper, we give a systematic, methodical, and formal approach to use scenarios in system and acceptance testing. Based on the environment-system model, we transform textual scenarios into requirements that are not only represented in a testable format but also contain testing steps (the relation of the environment) as well as the expected behaviour of the system (the relation of the system). We also unveil, in formal terms, the relationship between scenarios and testing. We use tabular expressions to represent the knowledge obtained from the formalisation of informal scenarios.

Keywords: *Requirements engineering, software testing, requirements knowledge representation, specification-based testing, relational methods, formal methods.*

1 Introduction

Requirements specification is the first level of specification reflecting users' expectations and needs. If it is done properly, it is also the closest specification to the users' intentions. Divergence from the users' requirements may be introduced into specifications in any phase along the development process down to the detailed design specification. Therefore, system and acceptance testing, which test the software against its requirements specification, play a vital

role in testing.

One of the most important purposes of testing is to increase the confidence of the user/client/stakeholder in a software system and to show, by demonstrating the system on chosen test cases, that the system behaves as it is intended. Testing from specifications offers testers and users confidence in the intended functionality of the system. Specification-based testing is currently immature. The scarcity of practical techniques and automated tools makes it tempting to address.

System and acceptance testing, despite their importance, are probably the least understood specification-based testing phases. In [2](1981), Celentano et al. wrote that there are no clear guidelines which precisely identify what these phases are expected to accomplish, how they can be done methodically, and how their effectiveness can be evaluated. Today, more than two decades later, there is no significant progress related to this issue. A clear-defined criteria that can guide testers through system and acceptance testing is greatly needed.

The term testing is often used in a much broader sense in the literature [12], by which it includes verification of specifications, static inspection and walk-through of code, and symbolic execution. We use it in a rather narrow, but most acknowledged sense. Software testing is the process of executing a system on test cases in order to determine whether the results it produces are correct as specified.

1.1 Scenarios and their role in testing

In system requirements and human computer interaction, the use of examples, scenes, textual descriptions and prototypes have attracted considerable attention [17]. All these

approaches can be called *use cases* or *scenarios* based approaches. *Use cases* or *scenarios* are, therefore, partial descriptions of user-system interactions. Scenarios enforce interdisciplinary learning, reduce complexity, and facilitate partial agreement. They describe a system from a user's perspective and focus on user-system interaction. The requirements analyst divides system requirements into several small scenarios, and captures one scenario at a time.

Scenarios can be used to:

1. Describe external system behaviour directly from the user's point of view. This supports early and continued user's involvement and interaction during requirements analysis [9].
2. Provide a decomposition of a system into functions from a user's perspective and that each such function can be treated separately. This is an application of the principle of separation of concerns.
3. Help validate the requirements specification: Decomposing the requirements into scenarios allows short feedback cycles between users and software engineers [7, 9].
4. Provide guidelines to build a cost-effective prototype [9].
5. Help improve modifiability [7]. Using scenarios leads to a decomposition of the system from a user's perspective. This makes it easy to deal with requirements evolution, thus improving modifiability.

Use cases or *scenarios* have become a powerful means for capturing requirements for software applications. Scenarios have been used both in research and industry [1, 17, 22]. The large variety of scenario based approaches emphasise a more user-oriented perspective in developing computer systems. Scenarios use also pervades industrial practice [22]. The CREWS research group has visited twelve projects in Germany and Switzerland that used scenarios in their software engineering process in one way or another [1]. The survey revealed that scenarios are flexible and broadly applicable approach. In [15, page 174], the authors write "Today, it is difficult to find Fortune 1000 IT departments where use cases are not used in some form or fashion."

In the literature, many authors advocate the use of scenarios for system and acceptance testing. The arguments given are articulated around the fact that scenarios describe how the users will be using the system. However, as far as we know, no author provided a clear technique to make a *systematic and formal use* of scenarios in testing. Indeed, in [9], the authors indicate, without exhibiting a systematic approach, that scenarios provide acceptance criteria for requirements-based testing. Ryser and Glinz [20] indicate

that the interaction sequences captured in a scenario are an ideal base for generating system test cases. In this paper, we give a systematic, methodical, and formal approach to use scenarios in system and acceptance testing. We also unveil, in formal terms, the relationship between scenarios and testing: the relation of the environment of the formal scenario is exactly the specification of the actions that the tester should perform to test the system against the considered requirements scenario.

1.2 Problem statement

Today, system and acceptance testing are performed informally without a systematic test-cases generation from system's requirements. The common practice today is to request the tester to read the requirements and "think" about some test cases. The testing activity is a very important activity and should not depend on the tester's background and expertise only. A more systematic and methodical approach to system and acceptance testing should be determined.

As indicated above, scenarios are partial description of the user-system interactions elicited directly from the user. By providing the scenarios, the user is indirectly providing the way the interactions with the system are intended to be. In this paper, we attempt to answer the following questions:

- How can the system be tested against the scenarios without going back to the users? The users already provided the requirements analyst with a description of their use of the system. There is no need to request them to use again their knowledge of the system in order to test it. For non-critical systems and when resources are limited, a testing budget is worthwhile being used to increase our confidence in the system's functionalities which the client is certainly going to use rather than in finding rare errors that might never be triggered when the system is actually used as specified by the scenarios.
- What would be the best way to use scenarios for testing?
- How to represent the knowledge obtained from a scenarios such that the system testing, based on that knowledge, can be automated?

In the next section, we present a short review of the literature as well as our solution to the problem. In Section 3, we give and discuss an example to illustrate our main points. In Section 4, we discuss and summarise the features of our approach, and we point to some future work.

2 Solutions

In the literature many pointed to the need for using scenarios for requirements-based testing [9, 20]. For instance, Ryser et al. [19, 20], motivated by the need for testing methods that support existing development methods, they propose a method for the derivation of test-cases for *system testing* purpose. The proposed method uses narrative scenarios formalised in statecharts [8].

Since tabular expressions are suitable for automated test-cases generation, we advocate their use to represent the knowledge obtained from the scenarios. We elaborate on this point in Section 4.

When we examine a textual scenario, we find that it reports on two kinds of actions: actions possibly performed by the environment (or user) and actions expected from the system as reaction to the environment (or *vice versa*). When the scenario is formalised, these two kinds of actions will be present in the formal specification of the scenario. In the Environment-System model [3], the scenario's specification is split into two disjoint parts: the part that concerns the system and the part that concerns its environment. It is the specification of the system that will be eventually built. Therefore, this part must be consistent with the other system parts which are derived from other scenarios. For more details on how a scenario is formalised and how to work out the specification of the system from the whole scenario's specification, we give in Section 3 an example (without going deeply into the details) and we refer the reader to [3, 5] where illustrative examples can be found as well as the mathematical model and its basis.

When a formal relational method is used [3, 5], the scenarios are formalised into *formal scenarios*. In this paper, a formal scenario, as introduced in [3], is a 3-tuple:¹ the space T of the scenario, a relation $R_e \subseteq T \times T$ describing the behaviour of the environment as described by the informal scenario, and lastly a relation $R_s \subseteq T \times T$ of the system as described by the informal scenario. The relation R on T such that $R = R_e \cup R_s$ is said to be the relation of the scenario. Also, R_e and R_s should satisfy $R_e \cap R_s = \emptyset$. For more details about formal scenarios, we refer the reader to [3, 5] where they are exhaustively introduced.

2.1 Tabular expressions as knowledge representation means

There are several ways to represent a single scenario. Jacobson [10] uses a mostly informal text notation. Rubin and Goldberg [18] introduce a tabular notation of scenario scripts. Hsia et al. [9] show that a scenario can be adequately represented by a regular language or, equivalently,

¹ $\subseteq, \times, \cup, \cap$, and \emptyset denote respectively subset, Cartesian product, union, intersection, and empty set.

$$Q \triangleq \bigcup_{i=1}^3 \bigcup_{j=1}^3 H_1[i] \cap H_2[j] \cap G[i, j]$$

		H_2		
		$\sqrt{y} < 27$	$\sqrt{y} > 27$	$y < 0$
H_1	$x = 3$	$x'^2 = x^2 + y^2$ $\wedge y' = y$	$x^2 = y^2$	true
	$x < 3$	$x'^2 = y^2$ $\wedge y' = y$	$x'^2 = x^2$ $\wedge y' = y$	false
	$x > 3$	$x'^2 = x^2$ $\wedge y' = y$	$x - x' > 3$ $\wedge y' - y < 3$	$x'^2 = x^2 + y^2$ $\wedge y' = y$
		G		

Figure 1. Predicate expression table with its interpretation Q

by a finite state automation. Glinz [6] advocates the use of statecharts [8]. Desharnais et al. [3] used relations and relational transition systems to represent formal scenarios. Khedri [13] and then Desharnais et al. [5] used tabular expressions (also known as SCR –Software Cost Reduction–tables) to represent and verify formal scenarios.

Each relation of the formal scenario is represented under a tabular format. Parnas in [16] gives the definitions of ten kinds of tables. The semantics of these kind of tables and more are given in [4, 11]. In this paper, to illustrate the use of tables, we use one class of table, called *predicate expression table* (also known as *normal relation table*).

The table in Figure 1 is an example of a two-dimensional predicate expression table. It has two headers, H_1 and H_2 , and a grid G . The entries of the headers are predicates that do not contain primed variables. The entries of the grid may contain both primed and unprimed variables. In the formulae that we use to specify a software system's requirements, the variables can occur primed or unprimed. Primed variables, like x' in the relation Q on the next page, are used to denote the values of the same components of the state after the operation (the ending state).

All entries of the table in Figure 1 can be interpreted as relations. For example, the third cell in the header H_2 can be interpreted as:

$$H_2[3] = \{((x, y), (x', y')) \mid y < 0\},$$

and the cell in the first row and the second column in the grid can be interpreted as:

$$G[1, 2] = \{((x, y), (x', y')) \mid x^2 = y^2\}.$$

The entries of H_2 play a role of pre-restriction in the interpretation Q . We put the interpretation Q on Figure 1 (to save space, we omit " $((x, y), (x', y')) \mid$ " in their definition expansions).

$$Q = \{ \begin{aligned} &(x = 3 \wedge \sqrt{y} < 27 \wedge x'^2 = x^2 + y^2 \wedge y' = y) \\ &\vee (x = 3 \wedge \sqrt{y} > 27 \wedge x^2 = y^2) \\ &\vee (x = 3 \wedge y < 0) \\ &\vee (x < 3 \wedge \sqrt{y} < 27 \wedge x'^2 = y^2 \wedge y' = y) \\ &\vee (x < 3 \wedge \sqrt{y} > 27 \wedge x'^2 = x^2 \wedge y' = y) \\ &\vee (x > 3 \wedge \sqrt{y} < 27 \wedge x'^2 = x^2 \wedge y' = y) \\ &\vee (x > 3 \wedge \sqrt{y} > 27 \wedge x - x' > 3 \wedge y' - y < 3) \\ &\vee (x > 3 \wedge y < 0 \wedge x'^2 = x^2 + y^2 \wedge y' = y) \end{aligned} \}$$

3 Illustrative example

In this section, we consider a simplified scenario “Open a bank account” from a banking system. Its formal specification as a formal scenario is composed of the scenario’s space, relation of the environment, and relation of the system. Due to space constraint, we omit the scenario space.

Open a bank account: *The system is in the initial state of B_Serv (Bank Representative Service) menu, prompting open or close an account options. The client comes in and asks to open an account. The bank representative selects “open an account”. The system changes to Open menu and a new account number is generated. The system then prompts for identification of the client. The representative may choose to go back, or enter the ID. If the representative chooses to go back, the system returns to the initial state of B_Serv. If the representative enters the ID, the system displays the new account number, adds the new account number into the set of accounts managed by the bank, registers which ID owns the account, and who is the new account owner. The system also adds the ID to the set of IDs it knows if it is not already in it. The representative finally chooses to go back. The system returns to the initial to R_Serv menu.*

A scenario is a partial description of the environment (user)/system interactions. It describes both the behaviour of the system to be built, as well as the behaviour of its environment. For instance, in the above scenario, the emphasised typeset text represents the environment behaviour. The text typeset in sans serif family font gives the expected behaviour of the system.

In general, and in all engineering fields, testing is placing the considered artifact in a controlled environment and to study its behaviour in that environment. For a software system, we find out that, when we are given the requirements under scenarios format, we have both a specification of the system as well as a specification of its environment. The

list of the tasks that the tester should perform are specified by the part of the scenario that concerns the environment. In the system-environment model that we use, the relation of the environment is the specification of the expected behaviour of the user. Therefore, it should be the specification of the expected behaviour of the tester. The relation of the system is the expected reaction of the system to the tester’s actions. There is no need to let the client test the system. The client already indicated how the system will be used.

Table of Figure 2 is obtained by formalising the part of the scenario that describes the actions of the environment of the banking system as described by the scenario “Open a bank account”. It also gives the behaviour of the tester in order to test the system against the considered scenario. The cells $H_1[1]$, $H_2[1]$, and $G[1, 1]$ of this table can be read as follows:

$$\begin{aligned} \text{Menu} &= B_Serv \wedge \text{Output} = \text{Open,close?} \\ &\wedge \text{InputBuffer}' = @Open \end{aligned}$$

The above predicate specifies that when the system is in menu B_Serv and when the output prompts *open* or *close* account options, then the tester should enter the command $@Open$ (a request to open an account). The predicates in the headers of the table of Figure 2 indicate the precondition to the tester’s actions. The predicates in the main grid indicate the actions the tester is requested to perform. For instance, the predicate contained in $H_1[1]$ and $H_2[1]$ (i.e., $\text{Menu} = B_Serv \wedge \text{Output} = \text{Open,close?}$) gives the precondition to the the action $\text{InputBuffer}' = @Open$ of the tester.

Table of Figure 3 gives the reaction of the system to the environment (tester) actions. For instance, to the action of the tester described in $G[1, 1]$ of the table of Figure 2, the system reacts by the actions described by the predicate of $G[1, 1]$ of the Table of Figure 3. If the system does not perform what is specified in $G[1, 1]$ of the Table of Figure 3, then the tester concludes that the test-case failed. In other terms, the system fails the test-case if the the tester gets a reaction of the system that satisfies the following predicate (which is equivalent to $\neg G[1, 1]$ of the Table of Figure 3):²

$$\begin{aligned} \text{Menu}' &\neq \text{Open} \vee \text{Output}' \neq \text{Id?} \\ \vee \text{NewAcc}' &\neq \text{New}() \vee \text{InputBuffer}' \neq \lambda \\ \vee \neg \text{OnlyChge}(m, o, nc, i) \end{aligned}$$

where $\text{OnlyChge}(m, o, nc, i)$ is a predicate which indicates that every variable x from the scenario’ space and other than m, o, nc , and i does not change (i.e., $x' = x$).

² \vee , \wedge , and \neg are Boolean operators and denote respectively **or**, **and**, and the negation. λ denotes the empty string.

$$Q \triangleq \bigcup_{i=1}^n \bigcup_{j=1}^m H_1[i] \cap H_2[j] \cap G[i, j]$$

		H_2		
		$\text{InputBuffer} = \lambda \wedge$		
H_1	Output = Open, close?	Menu = B_Serv	Menu = Open	Menu ≠ B_Serv ∧ Menu ≠ Open
	Output = Id?	InputBuffer' = @Open ∧ InputBuffer' ∈ A ⁺ ∧ InputBuffer' = @B_Serv	false	false
	Output = NewAcc	false	InputBuffer' = @B_Serv	false

G

Figure 2. Scenario's relation of the environment (Specification of the tester's behaviour)

To test the system against the scenario “Open a bank account”, the tester needs to perform the actions given in the table of Figure 2. Hence, we have a systematic way for testing against formal scenarios.

By formalising the scenario, we obtain requirements that are not only represented in a testable format but also contain the test steps (the relation of the environment) as well as the expected behaviour of the system (the relation of the system).

4 Discussion and future work

The formalisation model (i.e., environment-system model) used in the previous example demonstrated its suitability for automated verification of scenarios for inconsistency and for completeness [5, 14].

The relation of the environment, that the users of the environment-system model used to throw away and not make any use of it, is exactly a detailed testing specification. The use of tabular representation of relations to represent the knowledge obtained from the informal scenario are found to be very helpful for the automation of the requirements verification as well as for test cases generation.

It is not always convenient to test against individual scenarios. It is very often suitable to integrate the scenarios of a viewpoint to constitute a more complete view of the system and its environment. For that, we need to integrate all the relations of the environment of the viewpoint's scenarios to obtain a relation of the environment from the viewpoint's perspective. The integration of the relations of the environment can be performed automatically using the tool SCENATOR [14].

According to the source of information used to derive test cases, testing techniques may be classified into the following classes:

1. *Black-box* testing, also called *functional* or *specification-based* testing. In black-box testing,

$$Q \triangleq \bigcup_{i=1}^n \bigcup_{j=1}^m H_1[i] \cap H_2[j] \cap G[i, j]$$

		H_2		
		Menu = B_Serv	Menu = Open	Menu ≠ B_Serv ∧ Menu ≠ Open
H_1	Output = Open, close?	Menu = Open ∧ Output' = Id? ∧ NewAcc' = New() ∧ InputBuffer' = λ ∧ OnlyChge(m, o, n, i)	false	false
	Output = Id?	Menu = Open ∧ InputBuffer' = @B_Serv ∧ InputBuffer' ∈ A ⁺	false	false
	Output = NewAcc	Menu = Open ∧ InputBuffer' = @B_Serv ∧ InputBuffer' ∈ A ⁺	false	false
	Output = Id?	Menu = B_Serv ∧ Output' = Open, close? ∧ InputBuffer' = λ ∧ OnlyChge(m, o, i)	false	false
	Output = NewAcc	Menu = B_Serv ∧ Output' = Open, close? ∧ InputBuffer' = λ ∧ OnlyChge(m, o, i)	false	false
	Output = NewAcc	Menu = B_Serv ∧ Output' = Open, close? ∧ InputBuffer' = λ ∧ OnlyChge(m, o, i)	false	false

G

Figure 3. Scenario's relation of the system (Specification of the system's expected behaviour)

test cases are derived from the specification of the software, i.e., we do not consider implementation details.

2. *White-box* testing, also called *structural* or *program-based* testing. In this approach, we do consider the internal logical structure of the software in the derivation of test cases.

The classification of test techniques according to the criterion used to measure the adequacy of a test set leads to three main categories: coverage-based (e.g., control-flow coverage, data-flow coverage, coverage-based testing of requirements specifications, path-coverage, node-coverage, condition-coverage, etc.), fault-based, and error-based [21]. The technique proposed in this paper might be seen as a *Black-box* system testing technique since we do not assume any knowledge about the system's implementation.

Regrouping formal scenarios to obtain a testing work assignment is a task that needs further investigation. Our future work aims at establishing criteria for combining scenarios such that the obtained relation of the environment of a more global scenario is a more complete test specification. We need to investigate the conditions under which we can infer from a successful testing of a system against individual or groups of scenarios that the system is appropriately tested.

References

- [1] M. Arnold, M. Erdmann, M. Ginz, P. Haumer, R. Knoll, B. Paech, K. Pohl, J. Ryser, R. Studer, and K. Weidenhaupt. Survey on the scenario use in twelve selected industrial projects. Technical report, G.I. Working Group on Scenario-based Requirements Engineering, 1999.
- [2] A. Celentano and C. G. F. Liguori. A systematic approach to system and acceptance testing. In *Computer Program Testing*, pages 279–285. North-Holland Publishing, 1981.
- [3] J. Desharnais, M. Frappier, R. Khedri, and A. Mili. Integration of sequential scenarios. *IEEE Transactions on Software Engineering*, 24(9):695–708, September 1998.
- [4] J. Desharnais, R. Khedri, and A. Mili. *Interpretation of Tabular Expressions Using Arrays of Relations*, chapter 1, pages 3–14. Studies in Fuzziness and Soft Computing. Springer-Physica Verlag, 2001. Edited by Ewa Orłowska and Andrzej Szalas.
- [5] J. Desharnais, R. Khedri, and A. Mili. Representation, validation and integration of scenarios using tabular expressions. *Formal Methods in System Design*, 2003. Accepted for publication (Final version submitted on July 02, 2002).
- [6] M. Glinz. An integrated formal model of scenarios based on statecharts. In *Fifth European Software Engineering Conference*, volume 989 of *Lecture Notes in Computer Science*, pages 254–271. Springer, 1995.
- [7] M. Glinz. Improving the quality of requirements with scenarios. In *Proceedings of the Second World Congress for Software Quality (2WCSQ)*, September 2000.
- [8] D. Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Programming*, 8:231–274, 1987.
- [9] P. Hsia, J. Samuel, J. Gao, D. Kung, Y. Toyoshima, and C. Chen. Formal approach to scenario analysis. *IEEE Software*, 11(2):33–41, March 1994.
- [10] I. Jacobson. The use case construct in object-oriented software engineering. In J. M. Carroll, editor, *Scenario-Based Design: Envisioning Work and Technology in System Development*, pages 309–336. John Wiley and Sons, 1995.
- [11] R. Janicki and R. Khedri. On a formal semantics of tabular expressions. *Science of Computer Programming*, 39(1-2):189–213, March 2001.
- [12] C. Kaner, J. Falk, and H. Q. Nguyen. *Testing Computer Software*. Van Nostrand Reinhold, 1993.
- [13] R. Khedri. Sequential scenarios verification and integration using tabular expressions. CRL Report 374, Communications Research Laboratory, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada, June 1999.
- [14] R. Khedri, R. Wu, and B. Sanga. SCENATOR: a prototype tool for requirements inconsistency detection. In F. Wang and I. Lee, editors, *Proceedings of the 1st International Workshop on Automated Technology for Verification and Analysis*, pages 75–86, Taiwan, Republic of China, December 10–13 2003. National Taiwan University.
- [15] D. Kulak and E. Guiney. *Use Cases: Requirements in Context*. Addison-Wesley, second edition, 2004.
- [16] D. L. Parnas. Tabular representation of relations. CRL Report 260, Communications Research Laboratory, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada, October 1992.
- [17] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N. A. M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois, and P. Heymans. A proposal for scenario classification framework. *Requirements Engineering Journal*, 3(1), 1998. Also available as CREWS Report Series No. 96–01.
- [18] K. S. Rubin and A. Goldberg. Object behavior analysis. *Comm. ACM*, 35(9):48–62, September 1992.
- [19] J. Ryser and M. Glinz. SCENT: A method employing scenarios to systematically derive test cases for system test. Technical Report 2000/03, Institut für Informatik, Universität Zürich, 2000.
- [20] J. Ryser and M. Glinz. Using dependency charts to improve scenario-based testing. In *Proceedings of the 17th International Conference on Testing Computer Software (TCS2000)*. Washington D.C., June 2000.
- [21] H. van Vliet. *Software Engineering: Principles and Practice*. Wiley, second edition, 2000.
- [22] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer. Scenarios in system development current practice. *IEEE Software*, pages 34–45, March/April 1998.

Reuse of UML Class Diagrams Using Case-Based Composition

Paulo Gomes, Francisco C. Pereira, Paulo Carreiro, Paulo Paiva, Nuno Seco,
José L. Ferreira and Carlos Bento
CISUC - Centro de Informática e Sistemas da Universidade de Coimbra
Departamento de Engenharia Informática - Polo II
Universidade de Coimbra - Portugal
pgomes@dei.uc.pt

Abstract. In this paper, we present an approach to software design reuse based on Case-Based Composition. We show how this approach is integrated in an intelligent CASE tool being developed at the AI Lab of Coimbra. Two composition strategies are presented, along with experimental results.

1. Motivations

The complexity of software systems has increased along the past decades. Nowadays user interfaces are more sophisticated, data structures and system functionalities are more complex. Software development companies have to build new systems in less time and with limited resources. One possible solution to this situation is the reuse of software [4, 9]. There are several types of knowledge involved in the software development process that can be reused. System specifications, software designs, and code, are only a few. Code reuse has been the most common type of reuse, but it is not the most efficient. Decisions made at the design level have a much stronger influence in the system development than decisions made at the implementation level. This is one reason why we think that design reuse can be a solution for developing software faster and better. Designers need design reuse tools capable of helping them in this job.

Software design [3] is a cognitive task at a higher level of abstraction than code reuse. It deals with abstract concepts and it needs powerful reasoning capabilities. A CASE tool capable of providing assistance to the software designer should have several characteristics. It must be capable of understanding the user language, which is almost at the level of natural language. It must provide cognitive tools capable of complex reasoning abilities. It must learn new designs in order to reuse them later and to keep them updated. These are only some of the issues that a software design reuse system must address.

Software designers tend to reuse parts (or ideas) from different previous designs, integrating them into a coherent design. Most of the times the generated design is novel and bears characteristics that do not appear in the designs that originated it. From the cognitive point of view, design composition can be regarded as a cognitive process that can generate new designs. This process is a natural way of synthesizing new designs, and the quality of the output greatly depends on the designers experience. The more experience, the easier is for the designer to reuse previous designs, and to reuse them in a better way. We are interested in these two aspects of reuse: design composition and experience. This paper describes how we have modelled these aspects into a CASE tool.

2. Our Approach

Case-based reasoning (CBR) [7, 8] is a form of reasoning that uses experience in the form of cases to solve new problems. It enables the reuse of previously stored cases, seen as experiences, in new situations, which is the same cognitive process that designers often use in their profession. The main entity in CBR is a case, which represents a specific situation. In the design domain it can represent an artifact (a software design for example). Cases are stored and indexed in a case library. Usually associated with this case library there is an indexing structure, enabling a more efficient retrieval of cases from memory.

CBR comprises four main phases [1]: retrieve, reuse, revise and retain. The first phase is responsible for the search and retrieval of cases from the case library. Commonly this is done using an indexing structure that identifies the relevant cases for the target problem. Then, by applying a similarity metric, the retrieved cases are ranked. In the reuse phase, the CBR system modifies one or more retrieved cases adapting them to the target problem situation. Revising the solutions generated by the previous phase is the next step.

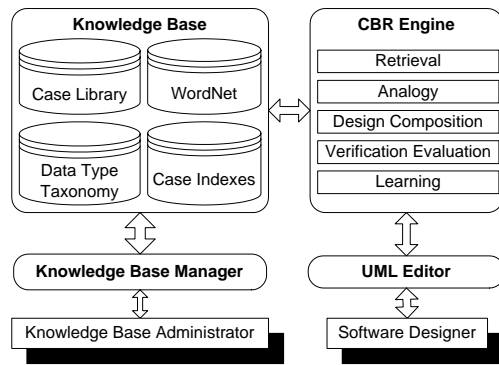


Figure 1. REBUILDER architecture.

Most of the times, this step is performed using domain specific heuristics or domain models. The last phase is the retaining (or learning) of the new generated solution in the form of a new case, thus closing the CBR cycle. This last phase allows the system to evolve in time, and to improve its performance.

The approach that we propose for an intelligent CASE tool is based on two key ideas: CBR as the reasoning framework for intelligent support, and the use of a general ontology as the conceptual basis for the knowledge used by CBR. Having these two issues in mind, we developed a system named REBUILDER that implements our approach.

REBUILDER is a Client-Server tool comprising two types of users: system administrator - responsible for the system's maintenance, and software engineer - uses the platform as a CASE tool. We selected the Unified Modelling Language (UML) [10] as the design language of our system. REBUILDER has four main parts (see figure 1): UML Editor, Knowledge Base (KB) that centralizes and stores all the system's knowledge, KB Manager that provides the administrator a way to maintain the KB, and CBR Engine that performs all the reasoning functions available in the UML Editor and the KB Manager. As for the general ontology we use WordNet, which is integrated in the KB.

The CBR engine performs all the inference work in REBUILDER. It comprises five sub modules:

Retrieval The retrieval module searches the case library for designs or design objects similar to the query diagram.

Analogy The analogy module maps designs from the case library, to the query design, resulting in a new diagram.

Design Composition The composition module can be used to adapt a past design (or part of it) to the query design using design composition.

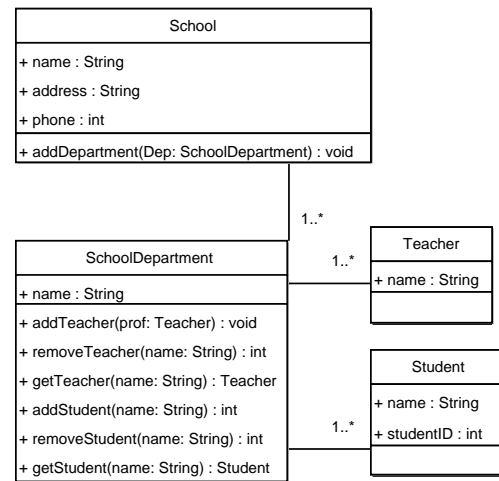


Figure 2. An example of a class diagram of an educational case.

Verification Evaluation The verification module checks the current design for inconsistencies, and evaluates designs.

Learning The learning module acquires new knowledge from the user interaction, or from the system reasoning.

This paper focus on the composition module.

The KB comprises a case library, the WordNet ontology, the case indexes and the data type taxonomy. In REBUILDER a case describes a software design in UML, through the use of Class Diagrams (figure 2 presents an example).

WordNet is used in REBUILDER as a common sense ontology. It uses a differential theory where concept meanings are represented by symbols that enable a theorist to distinguish among them. Symbols are words, and concept meanings are named synsets. A synset is a concept represented by one or more words. WordNet comprises a list of word synsets, and different semantic relations between synsets. The semantic relations between synsets, can be *is-a* relations (rat *is-a* mouse), *part-of* relations (door *part-of* house), and other relations. We use the word synset list and four semantic relation: *is-a*, *part-of*, *substance-of*, and *member-of*. Each diagram object has a synset associated, which is used for reasoning purposes.

Case indexes provide a way to access the relevant case parts for retrieval without having to read all the case files from disk. Each object in a case is used as an index. REBUILDER uses the associated synset of each object to index the case in WordNet. This way, REBUILDER can re-

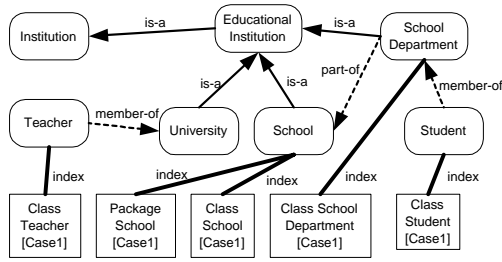


Figure 3. Part of the WordNet structure with object indexing from the example in figure 2.

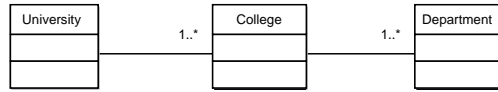


Figure 4. The class diagram of problem P1.

trieve a complete case, using the case root package, or it can retrieve only a subset of case objects, using the objects' indexes. As an example figure 3 shows the indexing of objects in class diagram of figure 2. Notice that School synset indexes two objects: the package of the class diagram, which corresponds to a school system; and the class representing the School object.

The data type taxonomy is a hierarchy of data types used in REBUILDER. Data types are used in the definition of attributes and methods. The data taxonomy is used to compute the conceptual distance between two data types.

The next section presents our approach to the reuse of UML diagrams using Case-Based Composition. Section 4 presents an example of design generation using this process. Section 5 describes some experimental results obtained from user interaction. Section 6 presents related work and some final conclusions.

3. Composition Module

The query used in the composition module is an UML class diagram, which is usually a small class diagram in its early stage of development (see figure 4). The goal of the composition module is to generate new diagrams that have the query objects. Generation of a new UML design using case-based composition involves two main steps: retrieving cases from the case library and using the retrieved cases (or parts of them) to build new UML diagrams. The two following sub sections describe these phases.

3.1. Retrieval of Diagrams

The retrieval process comprises two phases. In the first phase it uses the context synsets of the query diagram to get N objects from the case library, where N is the number of objects to be retrieved (N is user defined). This search is performed using the WordNet semantic relations that work like a conceptual graph, and case indexes that relate the case objects with WordNet synsets. The second phase ranks the set of retrieved objects using object similarity metrics.

The first phase uses the synset associated with the query object as an entry point in the WordNet graph. Then it gets the objects that are indexed by this synset using the case indexes. Only objects of the same type as the query are retrieved. For instance, if the query is a class, then only classes are retrieved. If the objects found do not reach N , then the search is expanded to the neighbour synsets navigating in the *is-a* relations. Then, the algorithm gets the new set of objects indexed by these synsets. If there are still not enough objects, the system keeps expanding until it reaches the desired number of objects, or till there are no more objects to expand.

Suppose that the N best objects are to be retrieved, $QObj$ is the query object, and $ObjectList$ is the universe of objects that can be retrieved (usually $ObjectList$ comprises all the library cases). The algorithm is:

1. $ObjsFound \leftarrow \emptyset$
2. $PSynset \leftarrow$ Get context synset of $QObj$
3. $PSynsets \leftarrow \{PSynset\}$
4. $ObjsExplored \leftarrow \emptyset$
5. WHILE ($\#ObjsFound < N$) AND ($PSynsets \neq \emptyset$) DO
 - (a) $Synset \leftarrow$ Remove first element of $PSynsets$
 - (b) $ObjsExplored \leftarrow ObjsExplored + Synset$
 - (c) $SubSynsets \leftarrow$ Get $Synset$ hyponyms (subordinates)
 - (d) $SuperSynsets \leftarrow$ Get $Synset$ hypernyms (superordinates)
 - (e) $SubSynsets \leftarrow SubSynsets - ObjsExplored - PSynsets$
 - (f) $SuperSynsets \leftarrow SuperSynsets - ObjsExplored - PSynsets$
 - (g) $PSynsets \leftarrow$ Add $SubSynsets$ to the end of $PSynsets$
 - (h) $PSynsets \leftarrow$ Add $SuperSynsets$ to the end of $PSynsets$
 - (i) $Objects \leftarrow$ Get all objects indexed by $Synset$
 - (j) $Objects \leftarrow Objects \cap ObjectList$
 - (k) $ObjsFound \leftarrow ObjsFound \cup Objects$
6. ENDWHILE
7. $ObjsFound \leftarrow$ Rank $ObjsFound$ by similarity
8. RETURN Select the first N elements from $ObjsFound$

The result of the previous phase is a set of N objects. The second phase ranks these objects by similarity with the query. Ranking is based on object similarity, and there are three types of object similarities: package similarity, class similarity, and interface similarity (see [6] for details on these metrics).

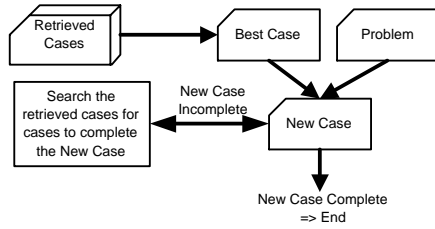


Figure 5. Best case composition strategy.

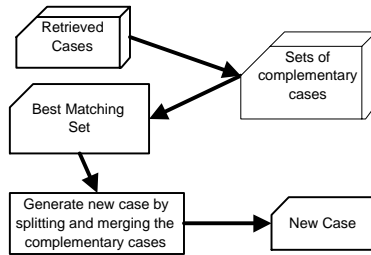


Figure 6. Best complementary cases composition strategy.

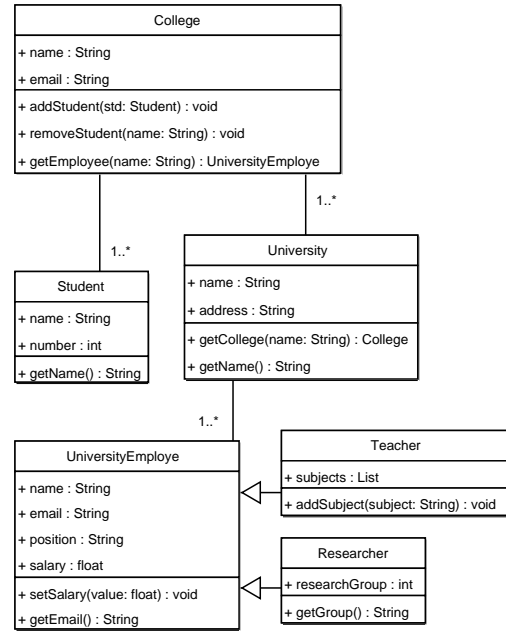


Figure 7. Class diagram of Case2 (a University).

3.2. Composition of Diagrams

After the retrieval of relevant cases, the composition module uses one or more retrieved cases to build a new case, using splitting and merging operations. Two adaptation strategies are used: best case composition, and best complementary cases composition.

In the best case composition (see figure 5), the adaptation module starts from the case most similar to the problem, mapping the case objects to the problem objects. The case mapped objects are copied to a new case. If this new case maps successfully all the problem objects, then the adaptation process ends. Otherwise it selects the retrieved case, which best complements the new case (in relation to the problem), and uses it to get the missing parts. This process continues while there are unmapped objects in the problem definition. Note that, if there are objects in an used case that are not in the problem, they can be transferred to the new case.

The best complementary cases composition (see figure 6) starts by matching each retrieved case to the problem, yielding a mapping between the case objects and the problem objects. This is used to determine the degree of problem coverage of each case, after which several sets of cases are constructed. These sets are based on the combined coverage of the problem, with the goal of finding sets of cases that globally map all the problem objects. The best match-

ing set is then used to generate a new case. These strategies are illustrated in the next section where an example is given.

4. Example

This example provides an illustration of the case composition mechanism of REBUILDER. Suppose that the class diagram of figure 4 is used as query diagram (problem P1). The algorithm starts by the package synset of P1, which is the synset corresponding to *University*. Suppose that the algorithm retrieves two cases: Case2, which corresponds to the class diagram of figure 7, and Case1 corresponding to the diagram of figure 2. These cases are then ranked by similarity to the problem, which gives Case2 a score of 0.7 and Case1 0.25.

The next step is to build a new case using case composition. We selected the best case composition strategy to generate the new case. The first thing that the algorithm does is to create a new case, which is a copy of P1. Case2 is selected due to its higher similarity with the problem. The design composition algorithm maps *University* and *College* from Case2 and P1, transferring the mapped case objects to the new case corresponding objects. Then the case objects that did not map are transferred to the new case only if mapped objects depend on them. After this, the problem has an unmapped object, which is *Department*. Case1 has

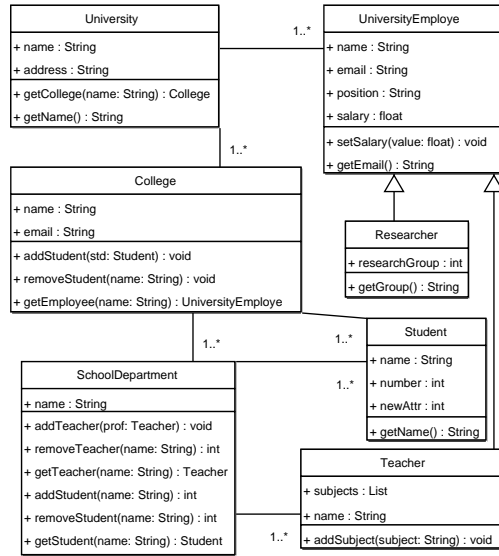


Figure 8. The solution generated by the design composition mechanism.

an object (*SchoolDepartment*) that has the same classification as *Department*, so it can be mapped completing the new case, resulting in the diagram of figure 8.

Note also that all additional objects from Case2 have been transferred to the new case because they depend on the mapped objects. Other aspect, is that the mapping of Case1 with the problem added two additional objects (*Student* and *Teacher* from Case1), which already existed in the new case (from Case2). These two additional objects were merged with the corresponding objects yielding the diagram of figure 8.

5. Experiments

In order to test the solutions generated by the design composition mechanism, we have performed user evaluation experience. In these experiences, we used a KB with 60 software designs, mainly about four different domains: banking information systems, health information systems, educational institution information systems, and store information systems (grocery stores, video stores, and others). Each design comprises a package, with 5 to 20 objects (total number of objects in the case base is 586). Each object has up to 20 attributes, and up to 20 methods. These designs are defined at a conceptual level, so the design is at an early stage of development having only the fundamental objects.

Four problems were defined, each one having one package with three objects (classes or interfaces), which were re-

Table 1. Experimental results obtained from test users.

	BCC	BCCC
Average number of objects by solution	11.25	8.13
Average number of relations by solution	11.00	7.63
Average number of incorrect objects by solution	1.98	1.04
Average number of incorrect relations by solution	2.73	1.69
Average of incorrect objects by total number of objects	17.6%	12.7%
Average of incorrect relations by total number of objects	24.9%	22.2%
Percentage of solution evaluated as incorrect	21.2%	17.3%

lated to each other by UML associations or generalizations. We defined one problem by domain of the Knowledge Base. For each problem REBUILDER generated four solutions, two using the Best Case Composition strategy, and the other two with the Best Complementary Cases Composition. The problems and their respective solutions were then presented to the test users (software designers and software engineers) for evaluation.

Eleven test users were inquired about each solution, giving their evaluation about the number of objects and relations that they considered inadequate or incorrectly defined, regarding the problem being modelled. Most of the designers made this judgment based on what they would delete from the suggested solution. The results obtained are presented in table 1 (BCC - Best Case Composition, BCCC - Best Complementary Cases Composition). These results show that solutions generated by the Best Complementary Cases strategy were considered more accurate, since the test subjects considered them with less incorrect objects and relations. One characteristic of this strategy that may be decisive for these results, is the number of objects and relations that the generated solutions have. As can be seen from table 1, the Best Complementary Cases strategy generates few objects and relations, but the ones that are selected are taken to be more relevant for the problem being solved.

To compare the computational performance of both strategies, we have tested the computation time that each strategy uses to produce solutions. We have used the same four problems and performed several runs with each one. We obtained the average of the computation time for the generation of one solution, using each strategy. Overall values are depicted in figure 9, and these figures show that the Best Complementary Cases strategy is best one, outperforming the Best Case strategy by 38%. These experiences were performed in a PC with Windows XP, an AMD XP 1500+ processor and with 512 Mb of RAM.

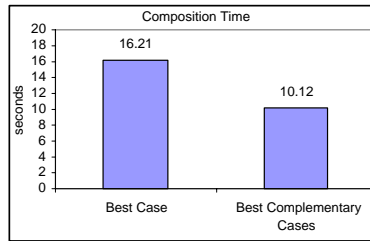


Figure 9. Average time results obtained for two generated solutions.

6. Conclusions and Future Work

Most of the research work on CBR for software reuse is on code reuse systems, which by definition have different characteristics from systems following our approach. Nevertheless there are some systems that can be considered related to REBUILDER. Gonzalez et. al. [5] presented a CBR approach for software reuse based on the reuse and design of Object-Oriented code. Cases represent three types of entities: classes, methods and programming recipes, thus allowing the retrieval of these types of objects. Deja Vu [11] is a CBR system for code generation and reuse using hierarchical CBR. Deja Vu uses a hierarchical case representation, indexing cases using functional features. Althoff and Tautz [2, 12] have a different approach to software reuse and design. Instead of reusing code, they reuse system requirements and associated software development knowledge.

Our approach to class diagram reuse using case-based composition enables REBUILDER to generate new diagrams using past designs. Most of the systems referred in the previous paragraph do not have the adaptation capability, which would enable the software designer to explore design suggestions. One main advantage of composition adaptation, is that the system has a broader scope of solution generation capabilities. Most of the adaptation techniques involving one case adaptation are limited to the transformation of the chosen case. From the test users evaluations and from the computational performance, it can be inferred that the Best Complementary Cases strategy performs better than the Best Case strategy in most of the problems used. Future work on this subject will focus on transference issues like, what kind of dependencies are transferable. Another important composition items are: merging objects and incoherence of objects. Both subjects can be addressed using WordNet as an ontology, providing semantics for reasoning.

Acknowledgments

This work was partially supported by POSI - Programa Operacional Sociedade de Informação of Fundação Portuguesa para a Ciência e Tecnologia and European Union FEDER, under contract POSI/33399/SRI/2000, by program PRAXIS XXI. REBUILDER homepage is <http://rebuilder.dei.uc.pt>.

References

- [1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.
- [2] K.-D. Althoff, A. Birk, C. G. v. Wangenheim, and C. Tautz. Case-based reasoning for experimental software engineering. Technical Report 063.97/E, Fraunhofer IESE, December 31, 1997 1997.
- [3] B. Boehm. *A Spiral Model of Software Development and Enhancement*. IEEE Press, 1988.
- [4] B. Coulange. *Software Reuse*. Springer Verlag, London, 1997.
- [5] C. Fernández-Chamizo, P. González-Calero, M. Gómez-Albarrán, and L. Hernández-Yáñez. Supporting object reuse through case-based reasoning. In I. Smith and B. Faltings, editors, *Third European Workshop on Case-Based Reasoning (EWCBR'96)*, volume 1168, pages 150–163, Lausanne, Suisse, 1996. Springer-Verlag.
- [6] P. Gomes, F. C. Pereira, P. Paiva, N. Seco, P. Carreiro, J. L. Ferreira, and C. Bento. Case retrieval of software designs using wordnet. In F. v. Harmelen, editor, *European Conference on Artificial Intelligence (ECAI'02)*, Lyon, France, 2002. IOS Press, Amsterdam.
- [7] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufman, 1993.
- [8] M. L. Maher, M. Balachandran, and D. Zhang. *Case-Based Reasoning in Design*. Lawrence Erlbaum Associates, 1995.
- [9] R. Prieto-Diaz. Status report: Software reusability. *IEEE Software*, 3(May), 1993.
- [10] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Reading, MA, 1998.
- [11] B. Smyth and P. Cunningham. Deja vu: A hierarchical case-based reasoning system for software design. In B. Neumann, editor, *10th European Conference on Artificial Intelligence (ECAI'92)*, Vienna, Austria, 1992. John Wiley and Sons.
- [12] C. Tautz and K.-D. Althoff. Using case-based reasoning for reusing software knowledge. In D. Leake and E. Plaza, editors, *International Conference on Case-Based Reasoning (ICCBR'97)*, pages 156–165, Providence, RI, USA, 1997. Springer-Verlag.

Reusing Knowledge on Software Quality for Developing Measurement Programs

Olga Jaufman
TU Kaiserslautern
P.O. Box 3049

D-67653 Kaiserslautern, Germany
D-67661 Kaiserslautern, Germany
+49 (6301) 707-253
freimut@iese.fhg.de

Bernd Freimut
Fraunhofer IESE
Sauerwiesen 6

Ioana Rus
Fraunhofer-Center Maryland
4321 Hartwick Road
College Park, MD, USA
+1 (301) 403-8971
irus@fc-md.umd.edu

Abstract. *In order develop high quality software, companies must understand what constitutes quality for their products and stakeholders and then manage and engineer quality accordingly. To support this, the measurement of software quality is essential. Various measurement approaches and quality models have been proposed for this, such as the ISO9126 standard, the models proposed by Boehm and McCall, or the GQM approach. The problem however is, that the application of these approaches implicitly assumes that the measurement personnel has experience in defining and measuring software quality, an assumption that is not true for many software companies. Therefore we propose to address this problem by supporting quality measurement with reuse of previously acquired knowledge. We gathered and organized quality measurement knowledge in a knowledge repository and designed a quality definition process that allows elicitation of quality requirements from multiple stakeholders.*

1. Introduction

Customer satisfaction progressively becomes a more important goal for software development companies who want to be successful and survive competition. Therefore these companies must improve their ability to understand what the customer means by “quality” and to develop software according to these requirements.

The challenge in doing so is that quality might not have the same meaning in the context of different application domains, different systems, or even for different stakeholders of the same system. Thus, there are multiple perspectives on quality that must be integrated for a system. For example, the customers or the users of a system have an “external” perspective, oriented more on the functionality of the system, and other execution properties such as performance, usability, and reliability. The developers on the other hand have an “internal” perspective, focused more on defects in software artefacts. Thus, there is a need for a common understanding and agreement between stakeholders of a

specific system on quality definition.

For being achieved, quality must be more than just measured in the final product; it must be properly defined, managed and engineered throughout development. For this purpose, quality indicators that can predict the final product quality must be defined and measured during development. This calls for the definition and selection of appropriate quality measures that require both application domain knowledge and software development knowledge. This can be a challenging task especially for people with less experience and knowledge in this area. The consequences can be firstly an incomplete characterization and measurement of the product quality and secondly a large amount of effort required to set and perform the measurement plan.

We propose to address this problem by supporting quality measurement with reuse of previously acquired knowledge. Some of the software quality knowledge is applicable across multiple systems, and is therefore transferable from one project to another.

We gathered and organized this knowledge in a knowledge repository called *software quality characterization and measurement* (SQCM) experience base, following the concept of *experience factory* (EF) [1]. The EF paradigm argues for reuse of experience collected from individual projects by analyzing, synthesizing and packaging this experience for future projects. The SQCM Experience Base captures generic knowledge and experience such as quality properties and corresponding measures, as well as when and how these measures should be taken and how to be used.

In this paper we describe the SQCM experience base, and the process of using it in support of measurement programs. The proposed concept was applied in a case study, the findings of which will be reported as well.

The target audience of this paper is envisioned to be project managers, software engineers, and process engineers, who are in the need for practical approaches for defining, documenting, and measuring software quality. Researchers can also benefit, by getting ideas about new and more efficient approaches for quality

modeling and measurement.

The remainder of the paper is organized as follows: Section 2 presents related work, and discusses similarities and differences between other approaches and ours. Section 3 describes in more detail the objectives of our research. Section 4 presents the experience base and the method for using it. The case study is discussed in Section 5. Section 6 concludes the paper with a summary and directions for future efforts.

2. Related Work

According to [4] there are principally two different ways to define software quality for a given product. Using a *fixed-model* approach it is assumed that all important quality properties are a subset of those in a defined model. To control and measure each quality property, relationships between measures, internal and external quality characteristics are used.

Examples of such an approach are the models proposed by McCall [7], Boehm [2], and ISO9126 [5]. In these models, key characteristics of quality are identified from a user perspective. These key characteristics are normally high-level external attributes (e.g., reliability, and maintainability). These high-level attributes are decomposed into lower-level attributes, which represent a refined view or internal view on quality. Measures are also proposed for these lower-level attributes.

The disadvantage of this approach is that the model's attributes are too general as they are supposed to be fitted for all contexts.

Therefore a different approach emerged, namely a *define-your-own-model* approach, as it was apparent that different contexts have different notions of quality. In these approaches, instead of using a pre-defined set of quality characteristics, a consensus on relevant quality characteristics is defined for a given product in co-operation with relevant stakeholders. These characteristics are then decomposed until measurable quality attributes are obtained.

Examples of a define-our-own-model approach are measurement approaches such as the GQM-approach [3] and the SQUID-approach [6]. The GQM-approach provides a framework for measurement programs, where relevant measurement goals for significant stakeholders are identified and then refined via questions into measures, through interviews with stakeholders.

The SQUID quality modelling approach [6] has the objective to define quality requirements for a particular product in a quantifiable manner. One basic idea of this approach is that a quality model has two components: (1) a structure model that defines model elements and their interactions, and (2) a content model that identifies a set of entities (e.g. attributes) linked in accordance with that structure. The structure model identifies, for example, that

quality properties are decomposed into sub-properties. Regarding the content model, [6] suggests to adopt one (or more) of the existing models and to adapt it to the organization.

The advantage of these approaches is that they can take into account the context in which quality is to be defined and can incorporate the views of the stakeholders. On the other hand, these models provide little guidance on how to define and elicit quality requirements.

Therefore, we aim at supporting these tasks by re-using experience about quality measurement.

3. Objectives

The purpose of our SQCM Experience Factory is to store and re-use experience related to the definition and measurement of quality, in order to support the measurement team in the process of setting up the quality definition for a product or project.

The challenge is, however, that the term "quality" is dependent on the domain, product, or even the stakeholders of a product (e.g., customer, or developer). Consequently, our experience base has to follow a *define-your-own-model* approach in order to be tailorable to the context in which it is to be used. The approach should allow abstract properties to be decomposed in more concrete sub-properties, specifically for each product and according to its stakeholders' views.

Yet, the implicit assumption with existing define-your-own approaches such as goal-oriented measurement is that the approach will be performed by people with measurement experience and knowledge. They should know what properties constitute product quality, how the relevant quality properties can be measured, when and by whom these properties can be measured, and finally how relevant it is to measure these properties. Unfortunately, not all people who are setting measurement programs have such measurement experience and knowledge.

We address this problem by providing an experience-based support that provides information about the properties that may be relevant to measure, how these properties can be measured, and for whom and when it may be relevant to measure them. In doing so, we are trying to build on top of previous work as much as possible.

The objective of the development of the SQCM Experience Factory is then to provide the support for initiating measurement programs more effectively and more efficiently. "More effectively" means that a measurement program covers all quality aspects that are relevant for product quality from project stakeholders' point of view. "More efficiently" means that less effort is required to set up a measurement program. The approach will also allow different stakeholders to communicate about quality needs using a common vocabulary.

4. The SQC-Experience Factory

4.1. The Experience Base

From the available define-your-own approaches, the SQUID-approach was appealing to us as its structure model already provides a sound scheme for an experience base. Therefore, we based the scheme of our experience-base on the SQUID structure model. The content of the experience base follows quality properties and their measurement models as proposed in the pre-defined quality models presented in the literature. In this section we discuss both the structure and the content of our experience base.

Structure

The structure of our EB scheme, adapted from [6], is shown in Figure 1

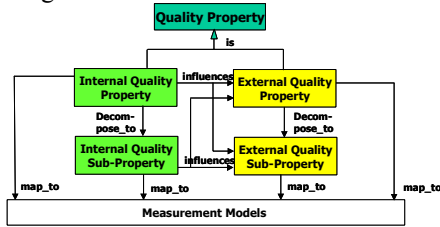


Figure 1 EB Scheme

Since we want to support both the customer-view of software as well as the developer view, we distinguish between internal and external quality properties [4]. A quality property can either be mapped directly to a measurement model that contains a descriptive model [3] according to which the quality property is to be operationally measured, or it is decomposed into more detailed sub-properties. This decomposition allows to organize quality properties hierarchically (e.g., like [7][2][5]) or flat such as within GQM.

In our pilot implementation we used a 4-level hierarchy, as we organized a large number of quality properties. However, in a different context where only a small number of quality attributes needs to be stored, a more flat implementation is also possible.

An important aspect of our objective is to measure quality during development and make inferences about final quality, since we want to evaluate the impact of new technologies on final quality as early as possible. The challenge is here to establish such relationships between internal and external quality by means of empirical (case) studies. Since these relationships tend to be dependent on a particular domain or even system, it is an important corporate asset to be captured in our experience base. Consequently, we aim at maintaining an influence relationship between internal and external quality.

In order to support and ease the selection of relevant quality properties, for each property we store the set of stakeholders and development phases, for which a

property might be relevant. Figure 2 shows an entry of our experience base.

Name of 3rd level property	Definition	Sub-property of	Applicable for object	Relevant for stakeholder	Relevant in phase
Reliability	The probability that the software will not cause a system failure for a specified time, under specified conditions.	Product Operability	Operational Software	User, Customer, Project Manager, Analyst, Designer, Programmer, Tester, Maintainer	Analysis, Design, Code, Test, Operation

Figure 2 Excerpt of EB

Each quality property is characterized by its name and a comprehensive definition as it was observed [10] that definitions within existing quality models are often terse and too short.

Content

Having defined the structure of our experience base, the next task is to populate it with an initial set of quality attributes. This can be achieved in several ways. First, it is possible for an organization to survey past measurement programmes and compile the set of quality properties under study as well as the measures used.

The second approach, which we followed, is to perform a survey of quality models published in the literature. Although these properties are typically not explicitly tailored to a specific domain or company, they can serve as a reasonable start.

Due to the abundance of quality models in the literature we faced several problems: Often quality properties are defined in too short and concise a manner, so that their exact meaning is fuzzy and unclear [10]. Moreover, when comparing models from different sources we observed inconsistent terminology about quality properties in the literature: Properties with the same meaning are denoted with different names, properties having the same names can differ in their meaning. Additionally, the published quality properties are on different layers of abstraction, for example, ranging from *Reliability* to *Mean Time Between Failures*.

Therefore, we carefully compared exiting definitions for quality properties and synthesized them in a hierarchy with 4-levels, where each quality property is explicitly defined. Additionally, we classified for each property the development phase and stakeholder, for which the property might be relevant, and identified applicable descriptive measurement models from the literature. To establish a relationship between internal and external quality properties, we adopted the indicators of *Reliability* identified in [7].

4.2. Using the Experience Base

The contents of the experience base are quality properties that have been or can be defined for products in the considered environment. In order to keep the contents up to date we defined processes that evaluate and update the contents as necessary. The focus of this paper is, however, the *Quality Definition Process*.

The objective of this process is to select and identify appropriate quality definitions and measurements for a given system by re-using such knowledge from the experience base. One key aspect here is the ability to consider and integrate different viewpoints on quality.

The basic idea of the process is to elicit which of the quality properties captured in the experience base are relevant for each stakeholder and to identify those quality properties that are important to several stakeholders or those that are only relevant for a small number of stakeholders. Figure 3 shows an overview of the process.

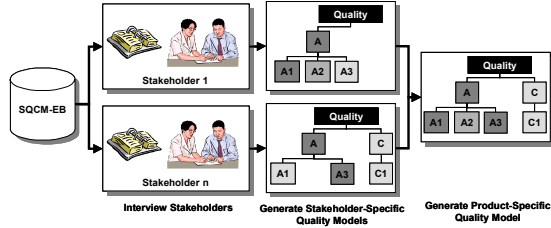


Figure 3 Quality Definition Process

The Quality Definition Process has the following steps: 1) Identify Stakeholders, 2) Interview Stakeholders, 3) Define a stakeholder-specific model for each stakeholder, and 4) Define a product-specific quality model by aggregating the stakeholders' views. In the following we will describe these steps in more detail:

The first step of this process is to identify those stakeholders that are to be considered in the quality definition of the system. Potential stakeholders are users, customers, project managers, contractors/developers, etc.

The second step is to interview individual representatives from the selected stakeholder groups. The objective of these interviews is to identify those quality properties that are important from the viewpoint of the stakeholder group. Therefore, structured interviews are conducted, in which each stakeholder rates the relevance of the quality properties in the experience base. To support these interviews, the interviewer derives a questionnaire from the experience base that includes those properties that might potentially be relevant for the stakeholder. For each quality property its name and definition is given, as well a 4-point-relevance scale. An excerpt from such a questionnaire is shown in Figure 4.

Question 18 (Level 2)	Portability The property "Portability" describes how easily it is to transport the software for use in another environments
	This quality property is for me ..
	<input type="checkbox"/> highly relevant <input type="checkbox"/> relevant <input type="checkbox"/> somewhat relevant <input type="checkbox"/> not relevant

Figure 4 Excerpt from questionnaire

In order to minimize the time required for these expert interviews, we exploited the tree structure of our experience base: If a quality property was rated as not relevant, quality properties in the hierarchy below that

property were considered as not relevant as well. Here, the questionnaire contained skip patterns that prevented these questions from being asked.

In order to detect whether the set derived from the experience base is complete, the interviewees are asked about missing properties that are to be considered.

In the third step a stakeholder-specific quality model is generated. When multiple persons per stakeholder group are to be interviewed, it is necessary to aggregate the individual viewpoints. Here it is possible [9] to perform a group interview and achieve consensus within the group or to aggregate individual answers mathematically.

The aggregated answers are visualized in a coloured quality tree (cf. Figure 5): the colours indicate whether quality properties have been rated as *very relevant*, *relevant*, or *somewhat relevant*. Quality Properties that were rated as not relevant are not included. Alternatively it is also possible to include only *very relevant* and *relevant* properties in the stakeholder-specific tree.

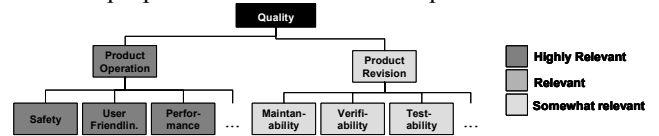


Figure 5 Role-Specific Quality Model (Example)

In the fourth step, a product-specific quality model is generated from the set of stakeholder-specific models. Applying mathematical aggregation it is possible to determine the relevance of a given quality property as the maximum relevance rating of all stakeholders, which implies equal importance of all stakeholders. In a more sophisticated computation, the importance of a group of stakeholders can be assigned a weight that will be considered in the final calculation. An alternative would be that ratings are assigned to the confidence in stakeholder's reply, according to his/her experience. Similar to the previous step, the resulting tree is visualized, as exemplified in Figure 6.

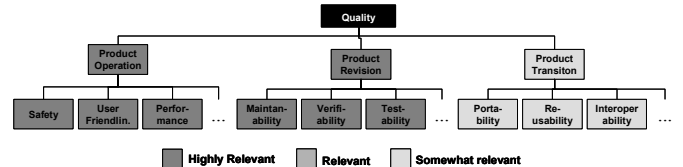


Figure 6 Product-Specific Quality Model (Example)

The advantage of these visualized models is that they clearly show which properties are important to many stakeholders. These are the properties that will be important to measure. In addition, an analysis of the role-specific models allows to make explicit the differences in quality perception from different stakeholders and to prioritize based on stakeholder importance.

Based on this model it is now possible to perform

focused GQM interviews or to directly define metrics. For both purposes, the experience base contains measurement models that will support these activities.

5. Case Study and Experiences

In order to evaluate our SQCM approach we performed a case study, in which we derived a product quality model for a concrete software product named *eWorkshop* (from “electronic workshop”).

The *eWorkshop* is a web-based product developed at the Fraunhofer Center for Experimental Software Engineering Maryland. The purpose of this software is to support virtual, on-line workshops with worldwide participants, and to capture the knowledge exchanged during each session for future analysis and dissemination. This software product has been used and has been under evolution for a couple of years.

5.1. Application of the Quality Definition Process

Since our initial experience base contained only quality properties from the literature we suspected that several product-specific quality properties were not included. Thus, we performed a first completeness check based on our understanding of the *eWorkshop*. Thereby we noticed several quality properties that one might expect to be important for such a system but that were not part of our experience base. Consequently we updated our experience base accordingly. This illustrates that it is an important aspect of the interviews to take the quality properties in the experience base as a sound start, but to be open for additional quality perceptions from each stakeholder. Following the Quality Definition Process described in Section 4.2, we then identified in the first step the set of stakeholders to be considered. For this product we identified the User and the Developer as stakeholders.

In order to prepare the interviews of the second step, we produced for each role a questionnaire containing those quality properties corresponding to the stakeholder (cf. Figure 4). Since we did not want to focus on a particular development phase, we considered all phases.

The interviews were held with one representative of the *Developer* role and two representatives of the *User* role. In the interview, the interviewees evaluated the relevance of the properties following the questionnaire. The duration of the interviews ranged between 15 and 25 minutes. We consider this time as rather short. Consequently, we regard such interviews as an inexpensive way to define an (initial) quality model.

Next, we generated the stakeholder-specific model for the Developer and the User based on the interview results according to Step 3 of the process. These models are shown in Figure 7 and 8. (Due to space constraints the

figures show only model excerpts.)

These models show that for the Developer quality properties related to product revision (e.g., maintainability, testability) play, naturally, a larger role than for the User. Also it can be observed that the Developer would prefer different measurement models than the User (Fault Density).

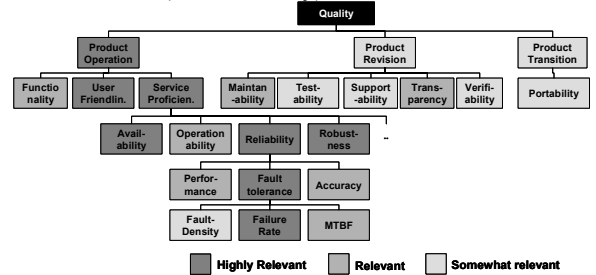


Figure 7 Developer View eWorkshop

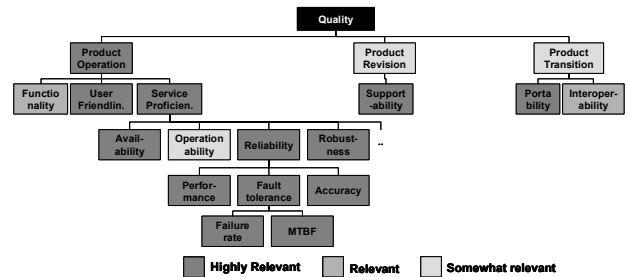


Figure 8 User View eWorkshop

Finally, these two models were aggregated using the maximum-relevance rating from each stakeholder-specific model as shown in Figure 9.

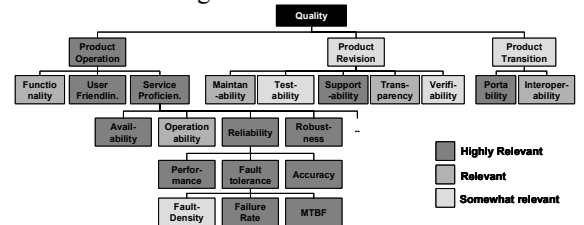


Figure 9: Unified eWorkshop Product Quality Model

These visualizations help to make different views explicit and allow a focused discussion on what to finally measure. The proposal for initial measurement models also helps the measurement team to prepare later steps.

5.2. Experiences on the Approach’s Usefulness

In order to evaluate the usefulness of our approach, we performed structured de-briefing interviews with each of the interviewees, to assess the approach’s pro’s and con’s. A questionnaire was prepared for these interviews and contained questions that tackled issues like the benefit of the approach, the perception of the time required to participate, the feasibility of the approach, and its

improvement opportunities.

Overall, the interviewees considered the approach as useful. Particularly, for the Developer it was interesting to see explicitly, which quality properties were highly relevant for the project manager. Thus, our approach clearly supported the communication about quality between different stakeholders. As overall strengths of the approach were regarded the usefulness for those Software Engineers who have little experience and knowledge in setting up software quality measurement programs and the usefulness for the characterization of a system for which it is not known what quality aspects may be relevant for the system's stakeholders. Moreover, the interviews can trigger the stakeholders to think about quality and to decide which quality attributes matter most to them.

As drawback was the length of the questionnaire considered. Here it would be useful to reduce the questionnaire using a more refined screening about potentially relevant quality properties, for example according to project environment characteristics. Additionally we think that a computer-based support of the questionnaire and experience base might ease the application as well.

Also a User representative said that the approach only gives "a pointer" to important quality attributes, which will have to be more refined and expressed in terms familiar to the project members.

Overall, we conclude from these results that our approach is useful and addresses the problem we wanted to tackle, namely the support of typically inexperienced personnel with an experience base as a preparation for more refined (GQM)-interviews.

Generally, the concept of our experience base can be easily transferred to contexts where quality elements can be structured as shown in Figure 1 or a subset thereof. The key idea of the Quality Definition Process, the relevance rating of proposed properties in interviews, is also easily transferable. However, the concrete structure of the questionnaire and the resulting stakeholder-specific and product-specific models might depend on the number of properties stored and depth of the hierarchy, which were both very high in our case.

6. Summary and Future Work

In this paper we proposed a practical approach to support quality measurement with reuse of previously acquired knowledge about the measurement of software quality. For this purpose we developed the SQCM experience base that contains knowledge in the form of quality properties and appropriate measurement models. We defined a quality definition process that allows to select and identify relevant quality properties for multiple stakeholders and thus facilitate communication about

different quality needs.

We reported our experience with this approach in a case study. Yet, the work we presented in this paper can be easily transferred into other organizations and enable them to re-use their own knowledge on measuring software quality.

The next steps with this approach are first to develop a tool that supports the tasks of deriving a stakeholder-specific questionnaire and that also supports the stakeholders in easily entering the information and automatically generating the model visualizations.

Also, although our approach helps to make explicit different views on quality, there is still the need to support the resolution of conflicts that result from those different views and to select those properties and measurements that should finally be measured.

7. Acknowledgements

This work was supported by the Otto A. Wipprecht Foundation, and in part by the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298. The authors thank their colleagues at the Fraunhofer Center Maryland, USA for their help in performing the case study.

References

- [1] V. Basili, G. Caldiera, and D. Rombach, "The Experience Factory," *Encyclopedia of Software Engineering - 2 Volume Set*, pp. 469-476, 1994.
- [2] B. Boehm, J. Brown, H. Kaspar, M. Lipow, G. MacLeod, and M. Merritt, *Characteristics of Software Quality*. North Holland Publishing Company, 1978.
- [3] L. C. Briand, C. Differding, and D. Rombach, *Practical Guidelines for Measurement-Based Process Improvement*, *Software Process Improvement and Practice Journal*, vol. 2, no. 3, 1997.
- [4] N. Fenton and S. Pfleeger, *Software Metrics - A Practical and Rigorous Approach*. Int. Thomson Computer Press, 1996.
- [5] ISO/IEC 9126 International Standard, *Software engineering – Product quality, Part 1: Quality model*, 2001.
- [6] B. Kitchenham, S. Linkman, A. Pasquini, and V. Nanni, *The SQUID-Approach to Defining a Quality Model*, *Software Quality Journal*, vol. 6, no. 3, pp. 211-233, 1997.
- [7] M. Li, C. Smidts, *Ranking Software Engineering Measures Related to Reliability Using Expert Opinion*, *Proceedings of ISSRE*, p. 246 – 258, 2000.
- [8] J.A. McCall, P.K. Richards, G.F. Walters, "Factors in Software Quality", *US Rome Air Development Center Reports RADC TR-77-369*, 1977.
- [9] M. Meyer and J. Booker, *Eliciting and Analyzing Expert Judgement: A Practical Guide.*, Academic Press, 1991.
- [10] D. Miller, *Choice and Application of a Software Quality Model*, in *Proceedings of the 10th International Conference on Software Quality*, 2000.

Reverse Engineering Software Architecture using Rough Clusters¹

J. H. Jahnke, Y. Bychkov
Department of Computer Science
University of Victoria
Victoria, BC, Canada
[jens|bychkov]@netlab.uvic.ca

Abstract - Software reverse engineering and program understanding deal with methods and techniques in support of maintenance and evolution of complex legacy software. A key challenge is to find effective mechanisms to (re-)create architectural abstractions of the software system, which aid human software engineers in understanding them. Much research has been devoted on developing algorithms for automated clustering of legacy software code into subsystem architectures. Still, few of these solutions are being used in industrial practice. We believe that this is mainly due to two main limitations, firstly, the lack of algorithms to represent approximate clusters, and secondly, the inability of clustering algorithms to use human expertise and domain knowledge about the legacy application. In this paper, we describe an approach that applies rough set theory for the purpose of legacy software clustering, in order to overcome these limitations.

I. INTRODUCTION

Today's software systems are large and complex. Much of the software that we use today has evolved over the course of many years and even decades. Often, such legacy systems are poorly documented. Their original architecture may have been eroded due to the many changes performed during their maintenance history. Up-to-date knowledge about their architectural design exists only in the minds of their developers. This knowledge is typically lost at the time when the developers leave the organization. The loss of this knowledge causes huge problems for the ongoing maintenance and evolution of many mission-critical legacy software systems. The sheer size of many legacy systems impedes the attempts of software engineers to gain understanding of the internal structure. As a result, making modifications to legacy systems is typically error-prone and costly.

Software reverse engineering has emerged as a discipline to address the problem of understanding and re-documenting legacy systems in order to provide a basis for their subsequent modification and evolution. In this context, many researchers and practitioners have investigated algorithms and methods to generate architectural abstractions for legacy software systems. Often these architectural abstractions are in the form of software subsystems, which represent clusters of tightly interdependent software artifacts. Such a partitioning facilitates human understanding of legacy code and makes it easier for a software engineer to determine the impact of modifications.

While great diversity of algorithms on how to compute software clusters has been developed over time [1], they are not well adopted in industrial practice so far. We believe that this is mainly due to two main limitations, firstly, *the lack of algorithms to represent approximate clusters*, and secondly, *the inability of clustering algorithms to use human expertise and domain knowledge about the legacy application*.

In collaboration with a Canadian tool vendor in the area of software reverse engineering, and funded by the National Science and Engineering Research Council (NSERC), we have developed an algorithm and tool prototype that applies rough set theory [2] for the purpose of legacy software clustering. In contrast to many other clustering algorithms [8], our approach is semi-interactive because it enables the user to inject his/her valuable domain knowledge. Moreover, our approach is iterative and incremental: rough software architectures can be refined semi-automatically based on user input.

II. SOFTWARE CLUSTERING

The first above-mentioned limitation of current clustering approaches (*"the lack of algorithms to represent approximate clusters"*) originates from practical experiences, which show that most (meaningful and useful) partitions of legacy software code artifacts into subsystems

¹ This work is partially supported by NSERC CRD Grant 239000-00 and klocwork Solutions Inc.

involve quantitative, uncertain clustering decisions. While for some software elements (e.g., classes, functions, variables etc.) the membership to a subsystem cluster can be decided unambiguously, other software elements might represent possible members of more than one subsystem. Many automatic clustering algorithms in software reverse engineering do not present such ambiguities to human software engineers, but they automatically choose the most “suitable” cluster, based on some internally computed quantitative measure. As a result, it has been argued that such algorithms often have stability problems, i.e., they violate the “rule of minimal change” (small changes in their parameterization or the subject legacy code should only result in small changes to the subsystem architecture produced [3]). A notable exception is Mitchell and Mancordis’ approach to evaluating the confidence in software clustering results [4]. They treat the clustering problem as a random-based optimization problem and propose to determine a confidence value for the goodness of clusters based in an analysis of multiple applications of this random-based algorithm.

Still, Mitchell and Mancordis approach is purely bottom up, i.e., it is only based on the legacy software code but does not consider input about the problem domain. However, experiences with real-world legacy applications show that the *meaningfulness* of clusters produced with these fully automatic mechanisms is limited. This is because intuitive *meaning* is typically associated with concepts in the *problem domain*, but not necessarily with concepts in the *solution domain*. Hence, we have to find ways of clustering artifacts in the solution domain guided by knowledge about concepts in the problem domain to better support human understanding of legacy software. These concepts would address the second limitation pointed out in the introduction, namely the *inability of clustering algorithms to use human expertise and domain knowledge about the legacy application*.

Baniassad and Murphy have addressed this second limitation with an approach that enables humans to specify *conceptual modules*, which are then subsequently used to cluster legacy code [5]. A similar approach using an architecture query language (AQL) has been developed by Sartipi and Kontogiannis [6]. A more visual approach based on polymetric views (lightweight software visualizations enriched with multiple software metrics) was proposed by Lanza and Ducasse [9]. While these and related approaches represent a significant step forward with respect to providing a more usable clustering of legacy code, they fall short on addressing the first problem, i.e., they do not represent approximate clusters. Approximation is unavoidable if clusters are generated based on a conceptual architecture defined by humans, because the implementation of an existing legacy system (most likely) will not completely reflect this architecture.

Therefore, the objective driving behind our research has been to develop a framework for clustering algorithms that supports approximation as well as the input of human domain expertise.

III. REPRESENTING APPROXIMATE SOFTWARE ARCHITECTURES

It is important to incorporate a concept of ambiguity and approximation in the clustering process. This enables the software engineer to judge the quality of any reverse engineered subsystem partitioning. However, the question is on how to represent “approximate” software architectures.

Mitchell and Mancordis, for example, have chosen a model that associates confidence values (percentages) with links between software artifacts in the same cluster [4]. While this approach addresses some of our concerns in principle, we believe that it is not very user friendly in practice: how should a user intuitively interpret an association between two artifacts in a cluster with confidence 56%? The authors try to address this problem by introducing the concept of a user-defined *confidence threshold*, which prevents software artifacts from being associated with a cluster by means of links with a confidence lower than this threshold. In this case, software artefacts that cannot be included into existing clusters with sufficient confidence are included in a newly created cluster of their own. Consequently, by decreasing or increasing the confidence threshold, software engineers can explore the software architecture using different levels of approximation. In practice, there are two practical problems with this approach:

- The reverse engineered architectural decomposition may change with each modification of the confidence threshold; a feature that is undesired if the decomposition is done according to a human-defined domain concept model.
- There is no way to represent *ambiguity*, e.g., the information that a software artifact might belong to cluster A *or* to cluster B.

One solution to the first problem is to pin-point clusters based on the selection of so-called seed artifacts (seeds for short). Seeds are software artifacts that, according to a human expert, clearly belong to a particular cluster as defined in the domain architecture. The relationships of seeds to other software artifacts are then evaluated to build the basis for automatic clustering. We are using this strategy in our approach described in the second part of this paper.

Addressing the second problem on how to represent *ambiguity* would require us to give up the constraint that each software artifact can be belong to one single cluster only. However, this would obviously cause problems if we wanted to represent non-ambiguous cases, e.g., situation

where a software engineer is certain about the membership of an artifact in a given cluster.

We propose to adopt rough set theory [2] to address this problem of representing ambiguity. Intuitively, a rough set \underline{S} uses two (traditional) sets $\underline{S} := (S_-, S^+)$ to approximate another set S . S_- is called *lower approximation* and contains only those elements that are known to belong to S , while $S^+ \supseteq S_-$ is called *upper approximation* and contains all elements that may belong to S . This model provides us with a simple, yet powerful way to address the ambiguity problem in software clustering: if we know with certainty that a software artifact belongs to a given cluster, we add it to the lower approximation of this cluster. Otherwise, if we are uncertain about the membership of an artifact in a cluster, we can add it to its upper approximation.

IV. ROUGH SET THEORY APPLIED TO SOFTWARE CLUSTERING

Software clustering algorithms commonly use an abstract model about the interdependencies among software artifacts, which has been extracted from the system to be reverse engineered. We use an *Artifact Dependency Graph* (ADG) for this purpose, which can be formalised as a tuple $ADG := (A, R, K, t)$, where

- A is a finite set of software artifacts,
- R finite multiset of dependencies among artifacts; elements in R are sets of form $\{a, b\}$, with $a, b \in A$,
- K is an alphabet of type labels,
- $t: A \cup R \rightarrow K$ is a labeling function, providing types for artifacts and dependencies (e.g., “class”, “variable”, “function”, “function call”, “uses” etc.).

It is the objective of clustering to find a (meaningful) partition of A . In traditional (crisp) clustering theory, a partition of A is a family $P = \{P_1, P_2, \dots, P_n\}$ of nonempty subsets of A such that each element in A is contained in exactly one element of P . The *ideal* result of a software clustering process would be a partition that fulfills the above constraint *and* is a meaningful abstraction for a human engineer. However, as argued earlier, these two requirements are often in mutual conflict, due to implementation idiosyncrasies, architectural erosion or other factors. Hence, there is the need to relax either of these requirements in practice. Most currently existing approaches have chosen to hold on to the traditional mathematical concept of crisp partitions, but have made compromises with respect to the *soft* requirement for meaningfulness.

We suggest approaching the problem from a different angle, i.e., to adopt a more relaxed mathematical notion of partitions in favour of attaining a more meaningful architectural abstractions. Using rough set theory, we can define a rough partition of A as a family $\underline{P} = \{\underline{P}_1, \underline{P}_2, \dots, \underline{P}_n\}$

of *strictly non-empty* rough sets approximating the ideal partition P .²

We can now relax the traditional (crisp) partitioning constraint (each element in A has to be contained in exactly one element in P) to the following (rough) partitioning constraint: each element in A either has to be contained exclusively in exactly one element in \underline{P} as a member of the element’s lower approximation, *or* the element has to appear in the upper approximation of at least one element in \underline{P} (without appearing in the lower approximation of any element in \underline{P}). This rough partitioning constraint can be formalized as:

$$(\forall a \in A) (\exists \underline{P}_i \in \underline{P}) ((a \in P_{i-} \wedge \neg (\exists \underline{P}_k \in \underline{P}) (P_{k-} \neq P_{i-} \wedge a \in P_{k-})) \vee (a \in P_{i+} \wedge \neg (\exists \underline{P}_k \in \underline{P}) (a \in P_{k-})))$$

The reader should note that this is not the only possible way to define the concept of a rough software partition. We could relax the above definition even further by changing our requirement for strictly non-empty partitions to non-empty partitions. The reason why we have not done this is based on our objective to generate meaningful clusters based on seed artifacts (cf. Section 3). We believe that clusters only have meaning if we can identify at least one seed artifact in the legacy system that belongs to it. Obviously, this seed artifact would appear in the lower approximation of the cluster, and, hence, the cluster would be considered strictly non-empty.

Fig. 1 illustrates a rough partition of an ADG that consists of three sample clusters. The lower approximation of each cluster is represented as an oval with solid background surrounded by a striped area representing their upper approximation. Artifacts are represented as circles, rendered with bold borders in case they represent seeds. The lines between artifacts illustrate their interdependencies. (We will show the role of these dependencies in the clustering process later.) The figure shows that rough clusters are allowed to overlap in their *boundary regions*. The boundary region $\square(\underline{S})$ of a rough cluster $\underline{S} := (S_-, S^+)$ is defined as $\square(\underline{S}) = S^+ - S_-$.

² A rough set $S := (S_-, S^+)$ is called *strictly non-empty* if and only if $S_- \neq \emptyset$, while S is called *non-empty* if and only if $S^+ \neq \emptyset$.

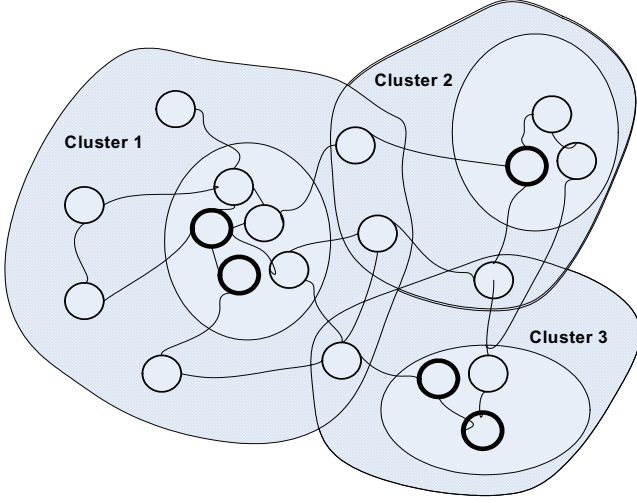


Fig 1. Rough Partition of an ADG

V. ASSESSING THE QUALITY OF ROUGH PARTITIONS

Obviously, it is desirable in reverse engineering to attain clusters with boundary regions as small as possible, in order to generate more concrete architectures. Based on the above theory, we can define a metric for the *concreteness* \diamond of a rough cluster \underline{S} to help a human to reverse engineer while assessing the quality of a rough cluster:

$$\diamond(\underline{S}) := \frac{|\underline{S}_{\approx}|}{|\underline{S}^*|}$$

We can further extend this metric for measuring the concreteness of a rough partition of an entire legacy software system as

$$\diamond^*(\underline{P}) := \frac{\sum_{\underline{P}_i \in \underline{P}} \diamond(\underline{P}_i)}{|\underline{P}|}$$

Intuitively, a situation in which software artifacts appear in several boundary regions expresses *ambiguity* with respect to the assignment of artifacts to the clusters defined in a partition. On the other hand, a situation in which artifacts appear in the boundary region of only one single cluster expresses a state of uncertainty about the *completeness* of a partition: Maybe clusters should be defined differently, e.g., by choosing other (or additional) seeds?

We define a metric for the degree of *ambiguity* (Θ) and the degree of *incompleteness* (Ξ) of a partition \underline{P} as

$$\Theta(\underline{P}) := \frac{|\{\underline{P}_i, \underline{P}_k, a \mid \underline{P}_i, \underline{P}_k \in \underline{P} \wedge \underline{P}_i \neq \underline{P}_k \wedge a \in \underline{P}_i \wedge a \in \underline{P}_k\}|}{2|\underline{A}| * |\underline{P}|^2}$$

$$\Xi(\underline{P}) := \frac{|\{a \mid \neg(\exists \underline{P}_i, \underline{P}_k \in \underline{P})(\underline{P}_i \neq \underline{P}_k \wedge a \in \underline{P}_i) \wedge a \in \underline{P}_k\}|}{|\underline{A}|}$$

All three metrics range from 0 to 1 (or 0-100%). A concreteness degree of $\diamond^*(\underline{P})$ of 100% is equivalent with an ideal (crisp) partition. In this case, the degrees of ambiguity $\Theta(\underline{P})$ and incompleteness $\Xi(\underline{P})$ equate to zero.

The concreteness of the partitioning in the example situation shown Fig. 1 is $\diamond^*(\underline{P})=51\%$, based on the average of the concreteness measures of the individual clusters $\diamond(\underline{P}_1)=42\%$, $\diamond(\underline{P}_2)=50\%$, and $\diamond(\underline{P}_3)=60\%$. The degree of ambiguity and incompleteness computes to $\Theta(\underline{P})=7\%$ and $\Xi(\underline{P})=21\%$, respectively.

VI. INCREMENTAL CLUSTERING PROCESS

In this section we propose an algorithm to put the above theory into use for software clustering and reverse engineering. We believe that user involvement is essential for reverse engineering meaningful software architectures from legacy code. Therefore, the clustering process suggested here has a semi-automatic and incremental nature. It consists of two main phases, namely *Concept Assignment* and *Partition Refinement* (cf. Fig. 2).

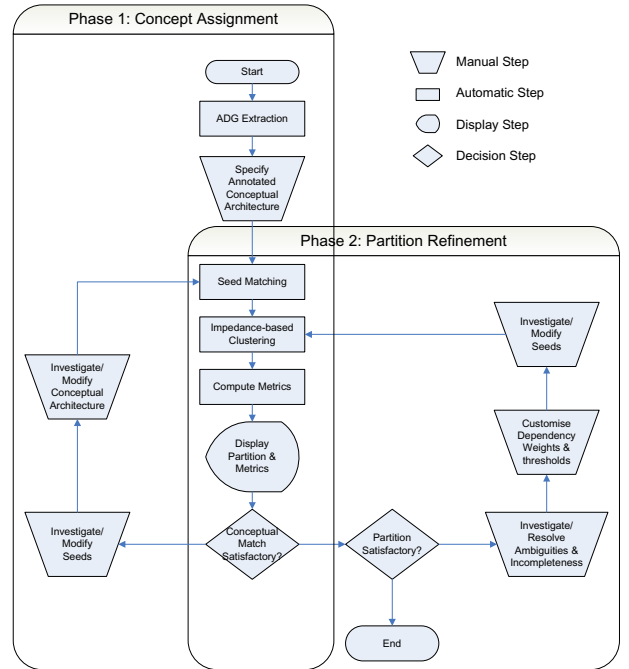


Fig 2. Incremental clustering process

The first phase is a top-down analysis of the software system, while the second phase is primarily a bottom-up analysis. The objective of Phase 1 (*Concept Assignment*) to come up with a first-cut rough partition of the subject software system based on user-defined conceptual model. The user defines a business-concept model (BCM) [7]

describing the problem domain implemented in the legacy software system. Furthermore, the user annotates each concept in the BCM with a number of keywords to further circumscribe it.

The next step (seed matching) is to find artifacts in the AGM that can be associated with the concepts in the BCM. This step is done automatically based on simple string pattern matching. Essentially, the keywords used to describe concepts in the BCM are matched against idioms used in the legacy system (represented by an AGM). Each concept defined in the BCM is taken as a rough cluster candidate, and each match for one of its keywords is taken as a seed candidate for this cluster. Now, our formal definition of a rough partition requires clusters to be strictly non-empty and seed artifacts to be exclusively owned by a single cluster (cf. Section 5). Therefore, at the end of the seed matching step, only those artifacts that have been matched as seed candidates exclusively for a single cluster are considered seeds for this cluster. Then, all cluster candidates without any seeds are deleted from the partition.

The following step (impedance-based clustering) evaluates the dependencies between seed artifacts and other artifacts in the AGM, in order to populate the rough clusters. For this purpose, we assign different weights to different types of dependencies in the AGM by defining a *weight function* $w:K \rightarrow]0, \infty[$. Dependencies with smaller weights (e.g., the use of a program variable) mean tighter coupling between software artifacts, whereas larger weights are associated to more loose dependencies (e.g., a function call). Our clustering algorithm evaluates the coupling of artifacts with seed artifacts by interpreting dependency weights as *impedances* analogously to impedances of resistors in electronics. Based on this idea, we define the impedance ω between two artifacts $(a, b \in A)$ in the $ADG: (A, R, K, t)$ as:

$$\omega(a, b) = \begin{cases} 0 & \text{if } a = b \\ \infty & \text{if } \neg \exists \{a, b\} \in R \\ \frac{1}{\sum_{r: \{b, a\} \in R} \frac{1}{w(t(r))}} & \text{otherwise} \end{cases}$$

The above definition of the impedance $\omega(a, b)$ equates to infinity if a and b are not directly connected in the ADG or identical. We can define the transitive impedance ω^* to account for transitive dependencies:

$$\omega^*(a, b) = \begin{cases} \omega(a, b) & \text{if } \exists \{a, b\} \in R \\ \frac{1}{\sum_{\{c, b\}, \{a, c\} \in R} \frac{1}{\omega^*(a, c) + \omega(c, b)}} & \text{else} \end{cases}$$

Now we can define the *cluster impedance* Ω between an artifact a and a given cluster \underline{P}_i as:

$$\Omega(\underline{P}_i, a) = \begin{cases} 0 & a \in X(\underline{P}_i) \\ \frac{1}{\sum_{s \in X(\underline{P}_i)} \frac{1}{\omega^*(s, a)}} & \text{otherwise} \end{cases}$$

The smaller the cluster impedance $\Omega(\underline{P}_i, a)$, the more artifact a belongs to cluster \underline{P}_i .

The user can parameterize the impedance-based clustering algorithm with two threshold values, namely the *lower threshold* (LT) and the *higher threshold* (HT). For each artifact a and for each cluster \underline{P}_i , the clustering algorithm compares $\Omega(\underline{P}_i, a)$ with HT and LT. The cluster assignment is done as follows: if the cluster impedance $\Omega(\underline{P}_i, a)$ between a cluster \underline{P}_i and an artifact a is smaller than LT and there is no other cluster for which this is true with respect to a , then a belongs to the lower approximation of \underline{P}_i . Otherwise, if the cluster impedance between \underline{P}_i and a is higher than UT, a is not a member of cluster \underline{P}_i . If none of the two conditions hold, a becomes a member of the boundary region of \underline{P}_i . Formally:

$$\begin{aligned} a \in P_{i*} & \text{ if } \Omega(\underline{P}_i, a) < LT \wedge (\neg \exists \underline{P}_k \in P)(\Omega(\underline{P}_k, a) < LT \wedge \underline{P}_i \neq \underline{P}_k) \\ a \notin \underline{P}_i & \text{ if } \Omega(\underline{P}_i, a) > UT \\ a \in P_i^* & \text{ else} \end{aligned}$$

When the impedance-based clustering step is finished, the quality metrics defined in Section 5 are computed and the result is displayed to the user. After investigating the rough partition, the user can decide whether the match between the BCM and the legacy system, as represented in the ADG, is satisfactory. It might take several iterations in Phase 1 (Conceptual Assignment) until the user is satisfied with the conceptual partitions matched to the BCM. During these iterations, the user can modify seed assignments and change the BCM to attain a better match.

The second phase of the interactive clustering process (*Partition Refinement*) starts when the user is satisfied with the result of the conceptual match achieved in Phase 1. In this second phase, the user can investigate and resolve ambiguities by moving artifacts from the boundary regions to the lower approximation of clusters. Furthermore, the user can resolve situations of incompleteness by creating new clusters and assign new seeds. The user can also customize the values for HT and LT, and the weights associated with the different dependency types. Like the first phase, the refinement phase is iterative and the user can re-evaluate the rough partition after each cycle of modifications. The process ends when the user is satisfied with the partition attained. (This does not necessarily have to be a crisp partition.)

VII. IMPLEMENTATION

We have implemented a prototype rough clustering tool based on the *Insight* reverse engineering tool produced by our industrial collaborator klocwork Solutions Inc. Among the various tools and functions provided by the rich Insight tool suite, our current prototype mainly uses Insight's robust, multi-lingual fact extractor, which can parse (legacy) software systems and generate the ADG. The ADG is maintained by Insight's software repository, which provides a powerful query interface. We made several small extensions to the repository schema in order to be able to represent rough clusters. We have not yet integrated our clustering mechanism with the advanced user interface of Insight. Rather, we have used GraphViz from AT&T labs as the prototype interface. Fig. 3 shows a screen shot of this tool, depicting two clusters. The status of artifacts (seeds, lower approximation, and boundary region) is visualized with different border colors (not visible in B&W print.) The user has the possibility to assign seeds, move artifacts between regions and change thresholds.

The BCM is currently defined in textual representation. Later, we will integrate the clustering tool with a graphical BCM modeling tool.

VIII. CONCLUSION

Software reverse engineering and program understanding are difficult problems. There seems to be an inherent tension between the desire to develop clustering algorithms that produce crisp and precise subsystem structures and the desire to produce *meaningful* structures. We have proposed an approach to relax the requirement for mathematically crisp partitions in order to attain more meaningful partitions. We believe that approximate partitions provide satisfactory answers for many questions

appearing during software reengineering activities. Moreover, rough partitions provide a mechanism to assess the quality of a legacy system architecture and plan software refactoring steps. We will now focus on evaluating our approach with practical case studies.

ACKNOWLEDGEMENT

The author would like to acknowledge Qilin Wang for his support and contribution to this research project.

REFERENCES

- [1] Koschke, R., Atomic Architectural Component Recovery for Program Understanding and Evolution, in Institute for Computer Science. 2000, University of Stuttgart: Stuttgart, Germany.
- [2] Pawlak, Z., Rough Sets. THEORY AND DECISION LIBRARY D: System Theory, Knowledge Engineering and Problem Solving. Vol. 9. 1992: Kluwer Academic Publishers.
- [3] Tzerpos, V. and R.C. Holt. On the Stability of Software Clustering Algorithms. in 8th International Workshop on Program Comprehension (IWPC'00). 2000. Limerick, Ireland: IEEE-CS.
- [4] Mitchell, B.S. and S. Mancoridis. CRAFT: A Framework for Evaluating Software Clustering Results in the Absence of Benchmark Decompositions. in Working Conference on Reverse Engineering (WCRE 01). 2001. Stuttgart, Germany: IEEE-CS.
- [5] Baniassad, E.L.A. and G.C. Murphy. Conceptual Module Querying for Software Reengineering. in Intl. Conference on Software Engineering. 1998. Kyoto, Japan: IEEE-CS.
- [6] Sartipi, K. and K. Kontogiannis. Component clustering based on maximal association. in Working Conference on Reverse Engineering (WCRE'01). 2001. Stuttgart, Germany: IEEE-CS.
- [7] Cheeseman and Daniels, UML Components: A Simple Process for Specifying Component-Based Software. 2002: Addison-Wesley.
- [8] Anquetil, N. and Lethbridge, T.C. Comparative study of clustering algorithms and abstract representations for software modularisation. in IEE Proceedings on Software Engineering, Vol.150, Iss.3, 24 June 2003
- [9] Lanza, M. and Ducasse, S., Polymetric views - a lightweight visual approach to reverse engineering. in IEEE Transactions on Software Engineering, Vol.29, Iss.9, Sept. 2003

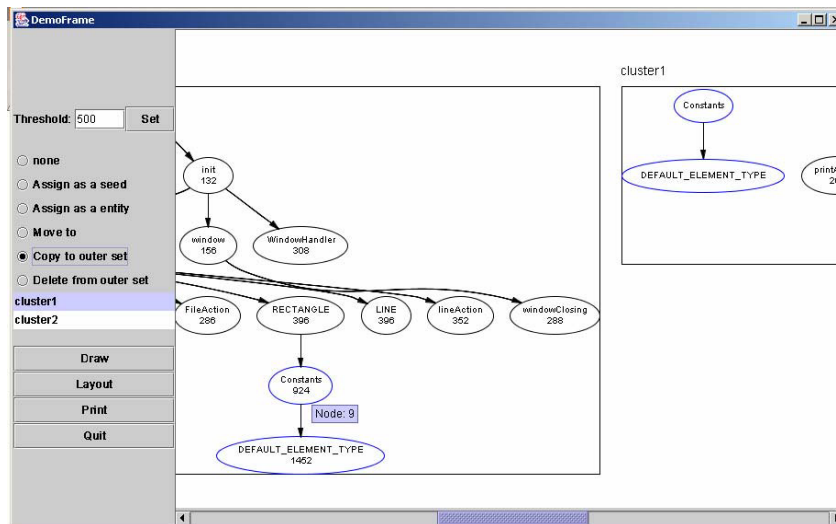


Fig 3. Rough Clustering Prototype

Software Architecture Modelling and Performance Analysis with Argo/MTE

Yuhong Cai¹, John Grundy^{1,2}, John Hosking¹ and Xiaoling Dai¹

*Department of Computer Science¹ and Department of Electrical and Computer Engineering²,
University of Auckland, Private Bag 92019, Auckland, New Zealand
{rainbow,john-g,john}@cs.auckland.ac.nz*

Abstract. We describe Argo/MTE, an extension of the open-source Argo/UML CASE tool that incorporates software architecture modelling facilities and performance test-bed code generation. We illustrate its application by example and explain the tool architecture and our experience using and evaluating it to date.

1. Introduction

Software architecture design and evaluation have become crucial in large scale systems development [4],[6],[8]. Validation of non-functional requirements is particularly critical and one of the most challenging of these to validate is system performance [6], [16], [17]. Existing architecture modelling and performance analysis tools are limited. Many modelling approaches have been taken, from informal visual design environments to formal architecture style specification and verification [5],[8],[13]. Performance analysis approaches range from simulation and rapid prototyping to reference benchmarks [4],[6],[14],[16],[20],[21]. Most have limitations when used on large-scale projects, such as scalability, integration with other development tools, result accuracy, and flexibility.

We describe an architecture design environment with performance analysis facilities which extends the Argo/UML open source CASE tool [18] to provide an integrated modelling environment. Several architecture modelling support features are added plus extensions of the XMI UML representation to capture architecture attributes. Performance analysis is based on test bed code generation where test code is synthesised, and performance tests run on real hardware and network infrastructure.

In the following, we provide a motivating example along with a survey of related research. We then overview and illustrate usage of the Argo/MTE architecture modelling and performance analysis environment. We briefly describe the tool's architecture and implementation, and our experience with the tool. We conclude with a summary and future research.

2. Motivation

Consider a complex architecture for internet micro-payment allowing many customers to buy information on the WWW on a pay-as-you-go basis, with many

small value transactions [2]. Fig. 1(a) shows an example of such a micro-payment system (NetPay) built using a component-based architecture [3].

When developing such software, architects must be able to model architecture, including many abstractions and their properties: clients, servers, machines, networks, protocols, caching, databases, messages, user interfaces etc in various levels of detail, from overview, refining into successively more detailed designs. Our interest is in how to support architects to gauge likely design performance, even from early, high-level designs [8]. Our approach focuses on generating executable code from architecture specifications and deploying this code on real hardware, to capture realistic timing information supporting incremental design refinement.

Many approaches have been used for performance estimation. Benchmarking [4],[6] uses reference architectures and load-testing simple implementations. Relative performances of different technologies used in reference implementations are compared. Benchmarks provide accurate measures for the benchmark application used, but are only a rough performance guide for related applications [6]. Rapid prototyping [11] develops partial software applications implementing performance-critical parts of the code e.g. network-centric and database-intensive. Much effort is often expended for even simple prototypes. If the architecture evolves prototypes must be modified and tests repeated, which is time-consuming and error-prone. Simulation approaches use models of distributed applications to estimate performance. Performance overhead estimates are based on architecture [1],[16] or middleware [12],[17] choices. As these approaches simulate performance, their accuracy varies widely and it is very difficult to obtain performance models for 3rd party applications such as databases.

3. Our Approach

In earlier work we developed a custom architecture modelling tool, retrofitting support for performance test-bed generation and analysis [7],[8]. Our new approach provides improved modelling and performance test-bed based analysis support within a standard CASE tool. This provides better integrated modelling and analysis support, uses existing model representation formats, and allows simpler refinement of architecture designs to OO

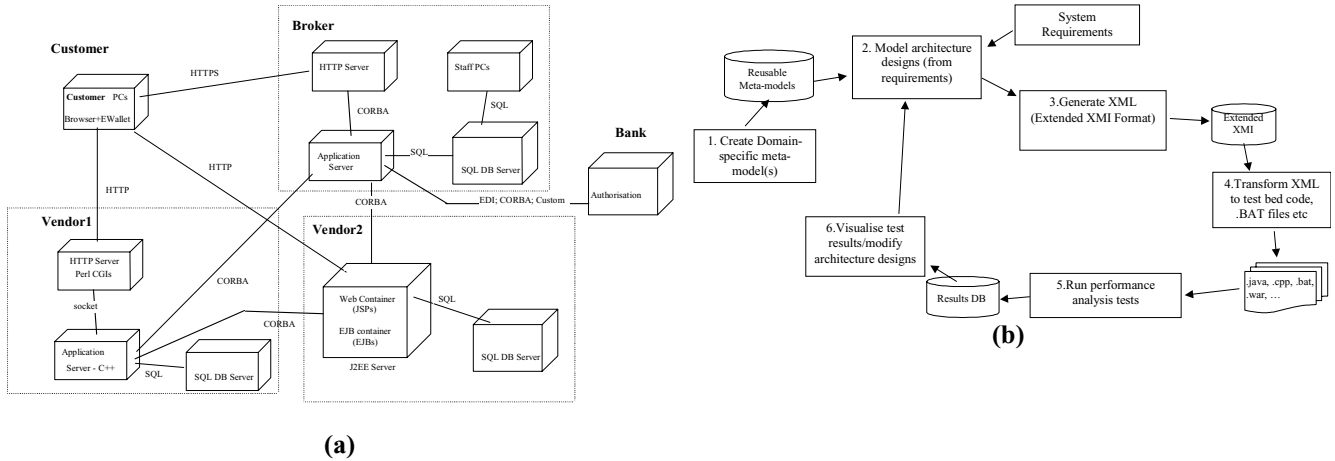


Fig. 1 (a)The NetPay micro-payment system architecture (b) Using the Argo/MTE Environment

designs and vice-versa. We chose to extend the Argo/UML CASE tool [18], [19] to develop Argo/MTE, but the approach is applicable to other modelling tools e.g. Rational Rose™, MS Visio™.

Fig. 1(b) shows how our environment is used by architects. (1) Multiple Argo/MTE domain-specific meta-models can be defined, each providing different modelling abstractions and code generators e.g. for web-based or real-time systems, etc. (2) Architecture models are developed using one or more meta-models and multiple design views. System requirements and specifications guide and constrain architecture design choices. (3) An extended XMI model format is used. (4) The model is transformed into files and scripts for code, compilation, database initialisation and deployment. (5) The generated test-bed code is compiled and deployed to multiple host machines and performance tests run. (6) Results are queried and visualised using various graphs which architects use to refine architecture designs and re-generate and run further performance tests.

4. An Overview of Argo/MTE Usage

We illustrate use of Argo/MTE using the NetPay architecture as an example. This is a complex architecture and here we consider only part of its design and one aspect of its performance. Fig. 3(a) shows Argo/MTE modelling an architecture meta-model i.e. a set of modelling abstractions for a particular domain. This example is a web-based enterprise system meta-model, including client, database and, application servers, remote object abstractions, and others. Argo/MTE uses Argo/UML view layout: menu and tool bars (1,2), tree view of model elements (3), diagram editing pane (4), and tabbed property sheet pane (5).

The architecture meta-model comprises element types (rectangular icons with names, stereotypes and properties), element type associations (solid lines),

hosting associations (dashed lines), and refinements (solid/dashed black line with one end point). Modelling elements define abstractions that can be composed in a model and their properties. An example of such types and properties is shown in Fig. 2. Associations specify how elements can be related, hosting associations specify how one element type relies on the existence of its host element, and refinements specify how one element type can be refined to a more detailed one to provide more information.

Element Type	Main Attributes	Property Description
Client	ClientType (AP, TP) Threads(TP)	Type of a client e.g. browser, CORBA client. Number of con-current clients run for tests.
RemoteRequest	RemoteServer (AP, TP) RemoteObject(AP, TP) RemoteMethod(AP, TP) RecordTime(TP) TimesToCall(TP) PauseBetweenCalls(TP)	Name of remote server to call The name of remote object The name of remote service Record time for this? Repetitions Pause duration between calls
AppServer	...	
RemoteService	...	
DBRequest	...	
DBTable	...	

Fig. 2. Meta-model type and attribute examples.

Architects choose one or more meta-models to use to create views of their architecture design. An Argo/MTE model view comprises elements (rectangles), element requests and services (labels), associations (solid black lines), message interactions (blue lines and highlights), hosting associations (dashed lines), and refinements (solid or dashed black line with one end point). Stereotypes indicate meta-model type correspondences. Each element has a property set derived from its meta-model abstraction. A high-level view for NetPay is shown in Fig. 3(a). NetPay comprises a customer PC-hosted browser and payment client (“E-wallet”) (1), a broker (2), and several vendor sites (3). The vendor here is a multi-tier architecture: the client browser accesses

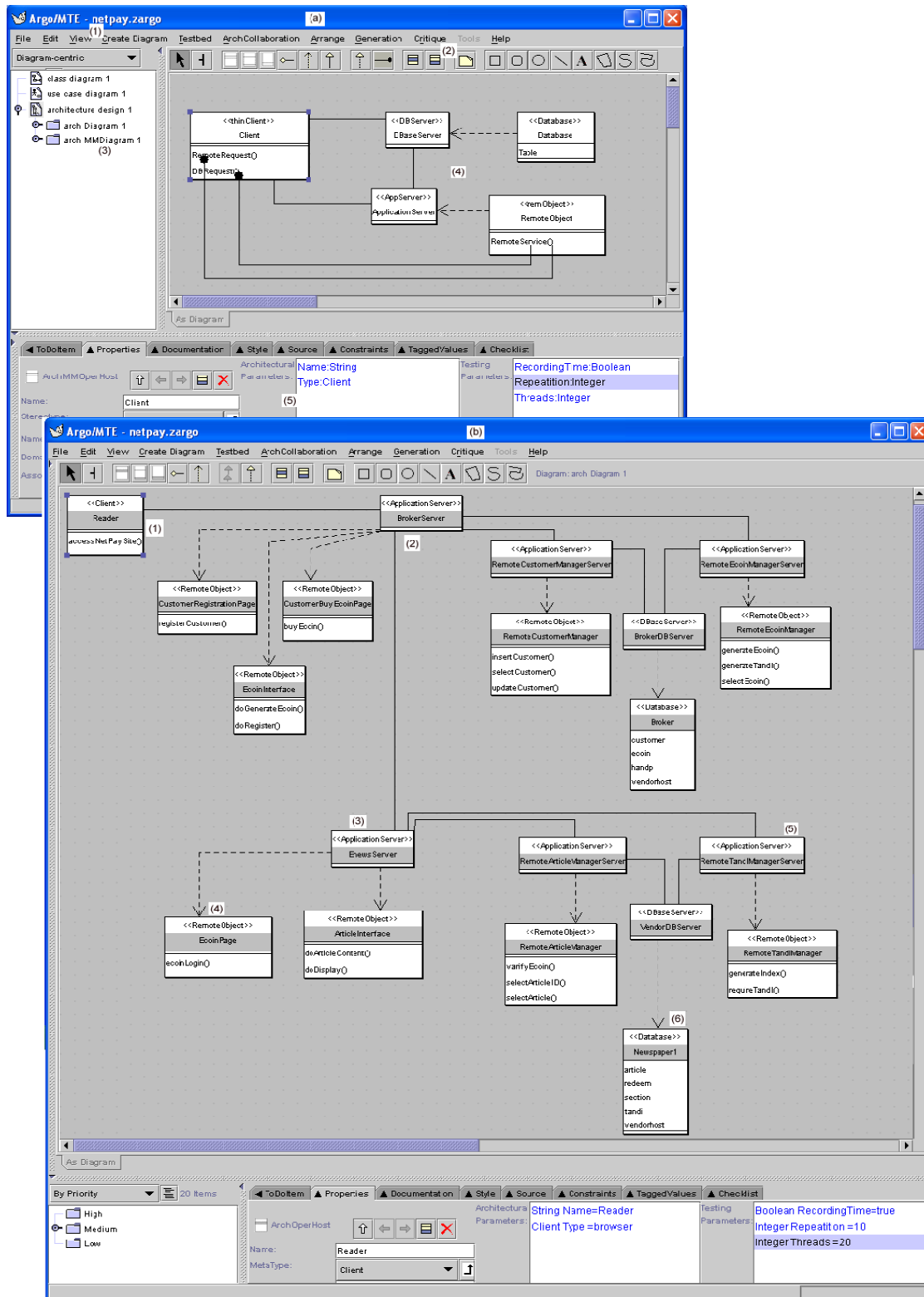


Fig. 3. (a) A domain-specific meta-model in Argo/MTE; (b) example architecture model in Argo/MTE.

web pages (4), which access application server components via CORBA (5), and a database (6). Each

abstraction links to other abstractions via relationships. Properties/parameters for <<Client>>Reader component

are at the bottom. Architectural parameters (AP in Fig. 2) support architecture modelling e.g. types and relationships. Testing parameters (TP in Fig. 2) support performance code generation, including number of client threads, timing information to record, number of request iterations, and pause between requests. We use a UML class icon-like architecture abstraction notation rather than UML deployment diagram shapes as we found the latter cumbersome and inflexible.

Multiple model views are supported for complex specifications. Fig. 4 shows three views of NetPay. Collaboration relationships between client requests and server services (1) visualise/specify message-passing relationships between elements. (2) shows just the message passing relationships between elements. Refinement of higher-level abstractions is shown in (3), where CustomerRegistrationPage service “registerCustomer()” is refined to constituent operations (each realised by business logic and database operations).

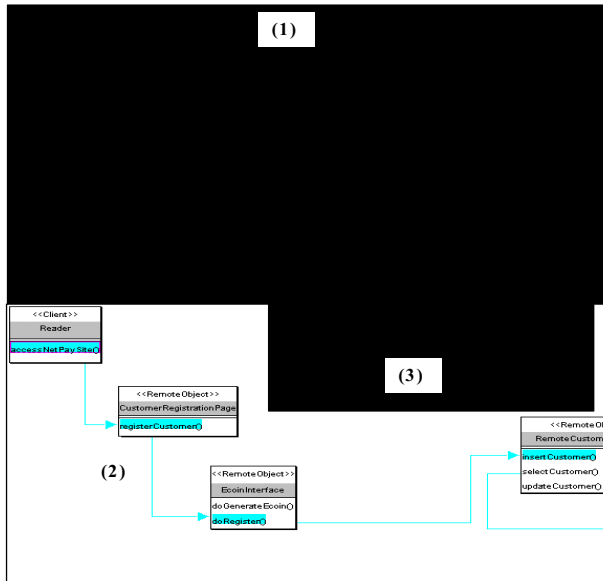


Fig. 4. Message associations in Argo/MTE designs and a simple refinement example.

Once an architect wants to assess performance of the modelled architecture, Argo/MTE generates test-beds and runs the tests. A basic assumption in our approach is that code in a component has minimal overhead, and hence performance is dominated by message passing etc through middleware and database access allowing a stub generation approach to still provide good performance data. Fig. 5 shows this process. An extended XMI format represents the design (1). XSLT scripts are run to generate Java, JSP, EJB, ASP and C# code files, and database initialisation, compilation and deployment script files (2). A deployment tool copies, installs, and runs these files on multiple client and server host

machines (3). Either thick-client testing applications are generated or Microsoft™ Application Centre Test scripts, used to run thin-client (web) tests. Performance information is captured in a database (4), which can be queried and graphed in various ways to compare results for different models and implementation parameters.

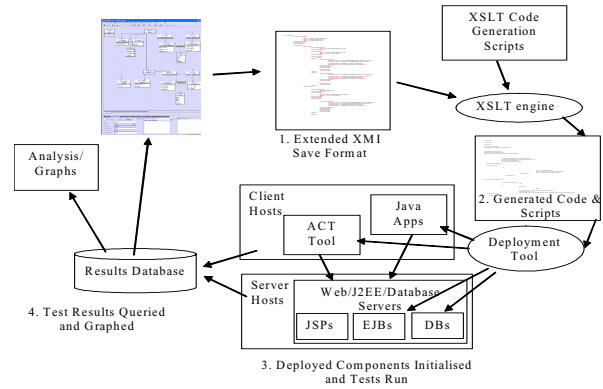


Fig. 5. Running, analysing and presenting results.

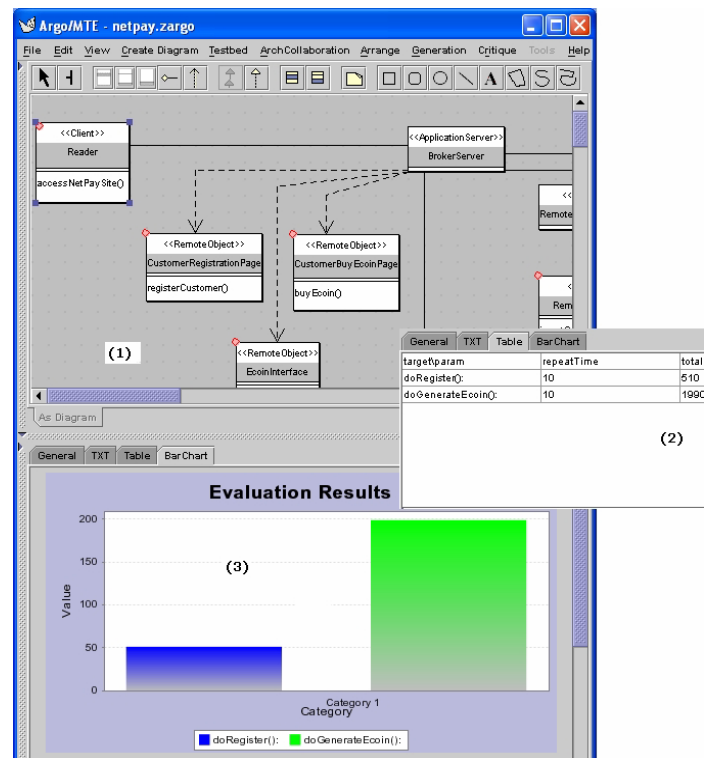


Fig. 6. Example performance analysis results.

Fig. 6 shows performance result presentation. Elements have a small circle at left top as a “result available” indicator. Fig. 6(1) shows several such elements, including “Reader” which has evaluation results displayed as a table (2) and bar chart (3). The table shows that each instance of “Reader” issues 10

requests each of tasks doRegister and doGenerateEcoin, taking 510ms (1990ms) to finish the requests, so on average it takes 51ms (199ms) to finish an individual task. The same results as a bar chart are in (3)

5. Design and Implementation

Fig. 7 shows key components of our extension of Argo/UML. A meta-modelling tool allows architects to define abstractions for different domains. The meta-model extends the existing Argo/UML XMI-based data representation. We chose to extend XMI as this was the approach used within Argo to represent models, but also to allow our saved architecture models to be partially read by other XMI-capable tools. Modelling tools were developed by specialising the Argo/UML class and collaboration diagramming tools.

The Xalan XSLT engine generates code and scripts. We modified a previously developed deployment tool to upload generated files to remote hosts and provide test co-ordination. Generated code captures timing data and stores this in a Microsoft™ Access database. MS Access forms and reports support test database browsing and visualisation. These facilities can readily be extended without modifying Argo/MTE itself. For some tests we generate thick-client applications to act as server invocation and data capture components. For thin-client systems, we generate configuration scripts for Microsoft™ Application Centre Test (ACT), which is instructed to carry out the tests and provide basic result visualisation, useful for load testing web applications.

6. Discussion and Conclusions

We have used Argo/MTE to model and test several software architectures and have compared generated performance results against that of actual implemented applications for accuracy. Applications modelled include several variants of thick and thin-client versions of an on-line video application [8], a Java Pet Shop application [15], substantial parts of NetPay [3], and several architectural approaches to an enterprise application integration (EAI) support system [9].

Argo/MTE successfully modelled these diverse architectures. The meta-modelling tool permitted us to define allowable modelling abstractions tailoring meta-models for thin-client and thick-client application modelling. We predominantly used the structural architecture modelling facilities to define clients and their requests, multi-tier servers, server objects, web components and relationships, and databases and tables. More complex architectures like the EAI and NetPay systems used multiple views with collaboration and sub-structural abstractions to manage the modelling complexity. Modelling abstractions of Argo/MTE were

mostly sufficient. Exceptions included complex, multi-element arguments to remote functions e.g. CORBA sequences and complex transactional logic e.g. multi-checkpoint transactions. Collaboration diagrams were useful for specifying dynamic behaviour but UML-style sequence diagrams would be useful to better capture operation sequencing.

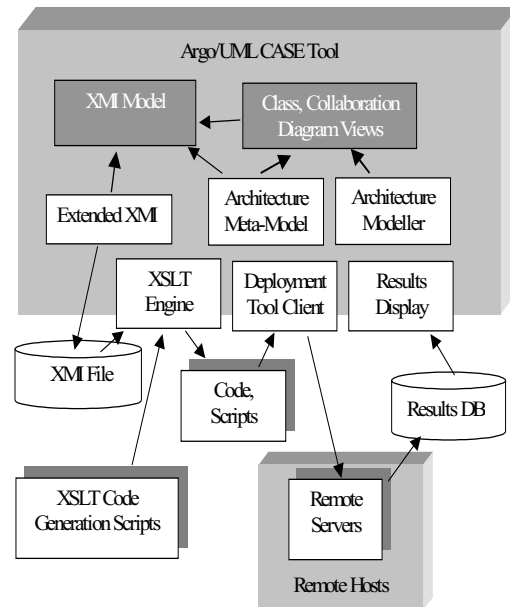


Fig. 7. The architecture of Argo/MTE.

We generated J2EE and .NET code for each system and performance tested applications using one or more SQL Server 2000 database servers. Some applications had pre-existing implementations in both J2EE and .NET (video system and Pet Shop), others had implementations in Java, J2EE, Java Messaging Service and CORBA (NetPay and the EAI application). We ran the same generated performance tests on the original, hand-implemented applications as were run on the generated test-beds. Some hand modification of these generated tests was needed to add correct argument values to properly drive hand-implemented servers. In general, performance results obtained from the generated test-bed code are accurate, with detailed Argo/MTE models producing performance results within 20-40% of the hand-implemented applications. Larger variances occurred with systems with complex business logic (conditional execution of substantial remote object and database services) and complex transaction processing logic as these violate our assumption of low overhead of such code. For some implementation technologies, including Java Messaging Service and .NET web services, we had only rudimentary code generators, resulting in inaccurate generated code. We also discovered implementation

deficiencies in the hand-implemented video and micro-payment systems which needed correction to sensibly compare their performance to the test-beds (a useful result in its own right). Our performance test database proved useful to capture all test results in one place and allow complex analysis and result visualisation.

Implementing and modifying XSLT code generators proved relatively time-consuming and improved support for this is needed. We envisage a small IDE within the tool to specify XSLT constructs and corresponding Argo/MTE extended XMI data, with ability to run parts of the code generator over test cases. The performance visualisation support is basic and needs improving. The XMI extensions are arbitrary, although they are a significant improvement on the proprietary architecture model format our previous work used. The format used may require revision as standardisation occurs in the representation of architecture information in UML and XMI. One final area for improvement is to permit users to specify ranges of values for testing parameters e.g. number of concurrent users and server threads. Ranges of averaged performance values could then be collected rather than a single average performance measure.

We have described extensions to a CASE tool for software architecture modelling and performance test bed generation. Argo/MTE provides graphical views for specifying performance test bed meta-models and architecture design diagrams stored as an extended XMI representation. This is used to generate a performance test bed, which, when run, produces relatively accurate performance results. We have demonstrated utility of the environment by modelling several architectures and favourably compared generated test-bed performance to that of hand-implemented versions of these systems.

References

- [1] Balsamo, S., Simeoni, M., Bernado, M. Combining Stochastic Process Algebras and Queuing Networks for Software Architecture analysis, *Proc 3rd Intl Wkshp Software & Performance*, 2002, ACM Press.
- [2] Dai, X. and Grundy, J.C. Customer perceptions of a thin-client micro-payment system: issues and experiences, *J. End User Computing*, 15, No. 4.
- [3] Dai, X. and Grundy, J.C. Architecture for a Component-based, Plug-in Micro-payment System, *Proc 5th Asia-Pacific Web Conference*, Sept 27-29 2003, Xi'an, China, LNCS 2642, pp. 251-262.
- [4] ECPerf Performance Benchmarks, August 2002, ecperf.theserverside.com/ecperf.
- [5] Gomaa, H., Menascé, D., and Kerschberg, L. A Software Architectural Design Method for Large-Scale Distributed Information Systems, *Distributed Systems Engineering J.*, Sept. 1996, IEE/BCS.
- [6] Gorton, I. And Liu, A. Evaluating Enterprise Java Bean Technology, In *Proc Software - Methods and Tools*, Wollongong, Australia, Nov 6-9 2000, IEEE.
- [7] Grundy, J.C. and Hosking, J.G. SoftArch: Tool support for integrated software architecture development, *IJSEKE*, Vol. 13(2), 2003., 125-152.
- [8] Grundy, J.C., Cai, Y. and Liu, A. Generation of Distributed System Test-beds from High-level Software Architecture Descriptions, *Proc 2001 IEEE Intl Conf on Automated Software Engineering*, San Diego, CA, Nov 26-29 2001.
- [9] Grundy, J.C., Bai, J., Blackham, J., Hosking, J.G. and Amor, R. An Architecture for Efficient, Flexible Enterprise System Integration, *PtoC 2003 Intl Conf on Internet Computing*, Las Vegas, June 23-26 2003, CSREA Press, pp. 350-356.
- [10] Grundy, J.C., Wei, Z., Nicolescu, R. and Cai, Y. An Environment for Automated Performance Evaluation of J2EE and ASP.NET Thin-client Architectures, *Proc 2004 Australian Conference on Software Engineering*, Melbourne, April 14-16 2004, IEEE CS Press.
- [11] Hu, L., Gorton, I. A performance prototyping approach to designing concurrent software architectures, In *Proc of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems*, IEEE, pp. 270 – 276.
- [12] Juiz, C., Puigjaner, R. Performance modelling of pools in soft real-time design architectures, *Simulation Practice & Theory*, 9, 2002, 215-40.
- [13] Kazman, R. Tool support for architecture analysis and design, In *Proc 2nd International Workshop on Software Architectures*, ACM Press, 94-97.
- [14] McCann, J.A., Manning, K.J. Tool to evaluate performance in distributed heterogeneous processing. *Proc 6th Euromicro Wkshp Parallel & Distributed Processing*, IEEE, 1998, 180-185.
- [15] MSDN, Using .NET to implement Sun Microsystem's Java Pet Store J2EE Blueprint application, October 2002, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/psimp.asp>.
- [16] Nimmagadda, S., Liyanarachchi, C., Gopinath, A., Niehaus, D. and Kaushal, A. Performance patterns: automated scenario based ORB performance evaluation, *Proc 5th USENIX Conf on OO Technologies & Systems*, USENIX, 1999, 15-28.
- [17] Petriu, D., Amer, H., Majumdar, S., Abdull-Fatah, I. Using analytic models for predicting middleware performance. In *Proc 2nd Intl Wkshp on Software and Performance*, ACM 2000, pp.189-94.
- [18] Robbins, J.E. and Redmiles, D.F. Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML, *Proc CoSET'99*, Los Angeles, May 1999, University of South Australia, pp. 61-70.
- [19] Robbins, J. Hilbert, D.M. and Redmiles, D.F. Extending design environments to software architecture design, *Automated Software Engineering*, vol. 5, No. 3, July 1998, 261-390.
- [20] Subraya, B.M., Subrahmanya, S.V. Object driven performance testing of Web applications, *Proc 1st Asia-Pacific Conf Quality Software*, IEEE, 17-26
- [21] Web Application Testing, WAPT Version 2.0, <http://www.loadtestingtool.com/>.

Software Project Risk Evaluation based on Specific and Systemic Risks

Hélio R. Costa^{1,2}, Márcio de O. Barros^{2,3}, Guilherme H. Travassos²
heliorcosta@infolink.com.br, marcio@cos.ufrj.br, ght@cos.ufrj.br

¹ CCA-RJ – Brazilian Air Force

Ponta do Galeão s/nº - Ilha do Governador – CEP: 21941-510 – Rio de Janeiro – Brasil

² COPPE / UFRJ – System Engineering and Computer Science Department
Caixa Postal: 68511 – CEP: 21945-970 – Rio de Janeiro – Brasil

³ UNIRIOTEC – Applied Computer Science Department
Av. Pasteur 458, Urca – CEP: 22290-240 – Rio de Janeiro – Brasil

Abstract

Recently, risk management has become a major concern for software project managers. Many studies have been conducted to identify risks that can influence the success of software projects. These studies were successful in defining a generic risk management process for software projects and developing techniques for risk identification, evaluation, planning, and control. However, we still lack techniques to compare projects risk levels and to evaluate risk diversification among several projects.

This paper presents an approach to evaluate a project overall risk level. It is based on the specific project characteristics and relative risks' importance, which can affect software projects. An extensive questionnaire is used to identify project-specific characteristics. An empirical study was planned and executed to quantify the relative importance of project risks for a specific system category. Observed results from the study are presented and discussed.

Keywords: software project management, software risk management, empirical studies.

1. Introduction

Software project risks can be classified in two groups: systemic risks and specific risks. Systemic risks are those factors that affect all software projects of a given category (such as information systems, military systems, off-the-shelf components, and so on). Any project in a category is subjected, to a specific level, to systemic risks that affect this category. Specific risks, on the other hand, are those factors associated to particular characteristics of a project.

The separation of risks in systemic and specific has its root in Economy. For instance, in the stock market every dealt stock has risks associated to the market and risks linked to peculiar aspects of an enterprise. Market risks (systemic) are due to the economic conditions of the country in which the enterprise resides, governmental influence upon markets, international economical scenario,

and global factors that influence all enterprises subjected to the same market. Enterprise risks (specific) are related to any particular organization, local market conditions, administration, reputation, and so on, i.e. while systemic risks evaluate the general scenario in which the enterprise is located, specific risks look within the enterprise for internal factors that can affect its performance.

In this paper we present an approach to evaluate a software project risk level based on its systemic and specific risks. This approach is based on a questionnaire built by aggregating taxonomies for risk identification presented in the software engineering literature. Each of its questions addresses a particular characteristic of a software project (specific risks). The questions are grouped in risk factors for capturing the project systemic aspects. By weighting the responses given by managers according to the relative importance of each risk factor, we calculate a project risk level.

The risk level and the weighted responses are useful for decision making in software projects. First, the risk level states how risky is a project in a single number, helping a manager to compare two or more projects based on their risk-and-return¹ ratios. Also, the weighted averages allow risk diversification among several projects. For instance, consider that the current projects developed by an organization may have a high analysis risk. The manager can decide to develop a new project, since its risk analysis is not so high, thus balancing the overall risk level among the risks that the organization is willing to accept.

The proposed risk level evaluation technique requires knowledge about the relative importance between risks in software projects. In order to determine this information, an

¹ The risk-and-return ratio is often used instead of ROI (return on investment) to evaluate an organization's willingness to develop a project. Unlike ROI, which only considers how much the organization will earn from the project, the risk-and-return ratio measures how much risk the organization will incur while attempting to earn the benefits from the project.

empirical study was planned and executed. The results from this study allowed us to determine, by collecting specialists' opinion, the weight of every risk for three distinct project sizes in a specific system category.

We organized this paper in five sections. Section 1 comprises this introduction. In Section 2, we present the technique that determines a project overall risk level. In Section 3, we describe an empirical study that was developed to quantify the relevance of software project risks for a specific system category, as required by the technique presented in Section 3. In Section 4, we compare the proposed approach to related works. Finally, in section 5 we present the main contributions of this paper and future perspectives of this research project.

2. Evaluating a Project Risk Level

To determine a software project risk level, the first step consists in answering a risk identification questionnaire. The questionnaire is based in the risk identification taxonomy presented in [1], which was complemented by other risk questionnaires and taxonomies presented in the literature [2; 3; 4; 5; 6; 7].

The questionnaire conveys 211 questions, which are classified in ten groups, named *factors*. Each question pertains to a single factor, as presented in Table 1. They aim to closely describe the abstract concept represented by the factor, relating it to more practical elements that can be evaluated by the manager from project characteristics. The questionnaire is extensive and will not be covered in this paper. Further details about its questions can be found at the URL <http://www.cos.ufrj.br/~heliorc/riskquest.html>.

Table 1- Distribution of questions among factors

<i>Factor</i>	<i># Questions</i>
Analysis	28
Design	17
Coding	11
Test	25
Planning	36
Control	17
Team	32
Policies and Structure	08
Contracts	21
Clients	16

The answer to each question is represented by a number from 0 to 5, where zero indicates that the issue covered by the question does not represent a risk for the project at hand (lower risk level) and 5 indicates that the question addresses an issue of major concern (higher risk level). The manager has also an option to assign the answer as not relevant (NR) for the project. Given these answers, the risk level evaluation technique proceeds in three additional steps.

In the next step, the manager calculates the average values of the answers for each risk factor. Questions

marked as not relevant are treated as missing values and are not considered in the average calculation. By using average, we implicitly assume that every question has the same relevance within a risk factor. This assumption is due to simplification, given the practical difficulties in evaluating the differences among so many (211) questions.

In the following step, the manager divides the average value calculated for each risk factor by 5 and multiplies the resulting number by the factor's adjustment value. Since the maximum value that can be attributed to the average value is 5, the division aims to normalize it, turning it into a number between 0 and 100%. In the last step, the manager sums the adjusted average values to determine the overall project risk level.

Table 2 presents an example of a project risk level calculation. In this example, two risk factors are considered. The first factor has 3 questions and a 70% adjustment value. The second factor conveys only 2 questions and a 30% adjustment value.

The risk factor adjustment value captures the systemic relevance of a risk factor and a process to determine this value is described in Section 3. However, some properties of this number should be stated:

- The sum of all risk factor adjustment values must be 100%;
- The higher the relevance of a risk factor, the higher should be its adjustment value.

The first property normalizes the project risk level, allowing it to assume any value between 0 and 100%. This is granted since each average answer can vary from zero to 5 and is divided, during the second step, by the maximum value that it can assume (5). The second property adjusts the specific risk evaluation (question answers) to the systemic risks presented by risk factors.

Table 2 – Calculating a project risk level

Answer the questionnaire	Factor 1			Factor 2	
	Q1	Q2	Q3	Q1	Q2
	2	4	3	2	4
Calculate average question answers	$\frac{(2 + 4 + 3)}{3} = 3$			$\frac{(4 + 2)}{2} = 3$	
Adjust average values	$\frac{3}{5} * 70\% = 42\%$			$\frac{3}{5} * 30\% = 18\%$	
Sum the adjusted average values	Risk Level = 60%				

The risk factor adjustment value is a delicate point in the proposed method. For instance, it should be stated that these values vary from project to project. We acknowledge this observation, but within a system category, this

difference can be attributed to specific risks. The later are captured in the answers to the questions of the taxonomy and reflected on the average values submitted for each risk factor. According to this logic, a high adjustment value for a factor that is not relevant in a project will have its final value reduced by the absence of risks or the attribution of low weights for its questions.

3. An Empirical Study for Risk Evaluation

Due to the many different types of software projects that can be undertaken for an increasing number of domains, it is supposed that the risk factors adjustment values required by the risk level evaluation technique presented in Section 2 can vary dramatically across different system categories. In order to determine these values for a particular category, we have planned and executed an empirical study.

With the purpose of reducing this research's scope and improve its precision, the first execution of the study was limited to evaluating the risk adjustment factors for Information Systems projects. However, in this section we present the study summary and suggest its application for other domains and system categories. The study was planned according to the methodology proposed by Wohlin et al. [8].

Objective: To determine the adjustment values which measure each risk factor's contribution to software project success. The adjustment factors reflect how critical a risk factor is for a specific system category. In this study, criticality is defined as the degree to which a factor contributes to the failure of a project.

Subjects: The study was performed with the assistance of professors, graduate students, and professionals with experience in software project development in the industry. The methods adopted to choose the subjects were *Quota Sampling* and *Convenience Sampling*. Thus, subjects were selected from distinct groups of the target population (software developers), but not randomly.

The subjects have given their opinion about the values of the risk factors in relation to the system category. Fifty (50) subjects were interviewed. All of them agreed to participate and signed in a consent form regarding the study. Seven (7) subjects hold a PhD degree in Computer Science, fifteen (15) have MSc degree, nineteen (19) have taken an MBA or equivalent course, and nine (9) are undergraduated professionals.

Among the subjects, eleven (11) are academic researchers and professors, while thirty-nine (39) currently work in software development organizations in Rio de Janeiro, Brazil. Twenty-six (26) different organizations were visited for the interviews.

Concerning their participation on software development projects, thirty-four (34) subjects have acted as project managers, fifteen (15) have worked as senior analysts, and one (1) subject have worked as senior programmer. Their

average experience in software development is about twelve (12) years, along which they have participated in about fourteen (14) projects, in average.

Project Size: while analyzing the risk factors, subjects were asked to keep in mind a specific project size. Subjects were also asked to choose the project size that closely represents their experience. The project size chosen by each subject was characterized by the effort required for its development (measured in man-months) and registered in a subject characterization form. Project sizes were used in data analysis for grouping subject opinions, so that distinct risk factors adjustment values could be determined for different project sizes.

Grouping: It was expected that subject experience and project size could influence the results of this research. So, we decided to block subjects and projects before data analysis. Subjects were classified, according to their experience, in three groups: low experience, medium experience, and large experience.

The grouping process was based on a characterization form that was filled by each subject before evaluating the risk factors. The form captured academic and industrial experience data about each subject. Examples of such information included: the number of years working with software development, the number of developed projects, academic formation, and degree of experience in risk management. This information allowed us to summarize subject's expertise in a single number (which varied from 0 to 15, the later representing the highest experience) and group subjects using the approach proposed by [9]. According to this classification, Table 3 presents the number of subjects and average weight in each group.

Table 3 – Subject count per group

	<i>Low Exp</i>	<i>Medium Exp</i>	<i>High Exp</i>
# Subjects	13	28	9
Avg. Weight	4.09	6.62	10.78

By grouping subjects, we were able to determine a common weight for the opinions given by subjects of the same group. We have assigned a weight of 1.0 for low experienced subjects. The weights of the following groups were determined by dividing their weight average value by the original weight average calculated for the low experienced subjects group. This process resulted in the weights presented in Table 4.

Table 4 – Subject groups' weights

	<i>Low Exp</i>	<i>Medium Exp</i>	<i>Large Exp</i>
Weight	1	1.61	2.63

Projects were classified as small, medium or large, according to the effort required for their development (measured in man-months). Projects developed with less

than 80 man-months were considered small; those developed with more than 80 and less the 250 man-months were considered medium; and those which required more than 250 man-months were considered large projects. According to this classification, Table 5 presents the number of projects and average size in each group.

No weights were assigned for projects. Instead, during data analysis they were adjusted by subject experience. The limits for the subject and project groups were established by minimizing the standard deviation among the participants of each group. This process was selected to enhance the similarity among elements of the same group.

Table 5 – Project count per group

	<i>Small</i>	<i>Medium</i>	<i>Large</i>
# Projects	17	18	15
Average Size	45.71	151.78	507.27

Instrumentation: As we needed to capture the individual experience of each subject, a subjective evaluation method was adopted. Among the available methods, the one chosen for this study was the *Paired Comparison*. In this method, the objects of interest are placed in a matrix so that each cell represents a comparison between a pair of objects. Only the elements above the matrix diagonal are considered. The subject analyses each pair of objects and determines which object shall receive the preference. In our study, the subjects were asked to subjectively determine the degree to which the preferred object was more important than the other one. After all comparisons, it is possible to determine which factor has the greatest relevance and successively until the least relevant factor.

The main advantage of comparing the objects in pairs is to reduce the evaluation complexity and improve the precision its result. Another point to be highlighted is that the human mind can more easily establish differences than estimate absolute values. Finally, through the comparison of an object with the others, the subject is forced to make a decision about the relation between two entities [10]. The disadvantage of this method is the great number of comparisons required in case of many objects: this value is in the order of the square of the number of objects.

The instrument chosen to support the comparisons was the MACBETH tool in its demonstration version² [11]. This tool allowed the subjects to formalize their preferences in a semantic way. By using linear programming, the votes were transformed by the tool in an interval scale which have their values expressed in terms of percentile. The MACBETH tool also checks for inconsistencies in the votes and helps in conflict resolution among comparisons, enforcing judgment coherence without influencing or restricting the freedom of the subjects in expressing their opinion.

² Even being a demonstration version, the tool has shown enough facilities to support the study.

Outlier Elimination: After collecting the judgment of all subjects, we have analyzed the data generated by the tool. The process started by eliminating outliers, where we applied elimination by standard deviation. In this process, we calculated the average and standard deviation for the votes given to each risk factor. Next, all the values that were separated from the average by two or more standard deviations were eliminated as outliers. From the 500 rough values obtained from subjects, only 14 were rejected. One value was eliminated for the *Coding*, *Control*, and *Contract* risk factors; two values were eliminated for the *Analysis*, *Design*, *Team*, and *Client* risk factors; and three values were eliminated for the *Planning* factor.

Data Analysis: The valid data were submitted to a T-test with a significance level of 0,05 (5%) and many comparisons were done in a statistical package in order to evaluate the differences and similarities between the averages of the collected values. The analysis was accomplished to the three project sizes (small, medium, and large). It was not observed any significant difference between these project sizes for the *Analysis*, *Design*, *Coding*, *Test*, *Planning*, and *Contract* factors. The remaining factors – *Team*, *Control*, *Policies/Structure*, and *Clients* – have presented a small, though significant, difference between small and large projects, but no difference between small/medium or medium/large projects. Therefore, it was decided to keep the values for all the factors and maintain the division of the projects in three sizes, despite the great resemblance observed in some factors.

The adjustment values obtained in the study after all the statistical analysis and normalization processes are presented in Table 6. They represent the degree of importance in a software development project.

Table 6 – Adjusted values of the factors

	<i>Small (%)</i>	<i>Medium (%)</i>	<i>Large (%)</i>
Analysis	12,36	12,57	10,78
Design	8,59	7,52	6,23
Coding	4,84	4,13	4,00
Test	7,36	6,17	5,82
Planning	15,26	13,04	13,85
Control	10,39	11,64	12,19
Team	11,28	11,53	12,63
Contracts	3,40	5,37	4,60
Policies/Structure	11,41	12,47	14,11
Clients	15,11	15,57	15,79

Internal Validity: Subjects were selected by *Quota Sampling* and *Convenience Sampling* based on their experience in software projects development. They were invited to take part of the study and were blocked in three groups. The number of subjects that voted for a certain project size was random and no mortality was observed, due to the nature of the study. The performance of

MACBETH tool was considered positive according to a qualitative questionnaire filled by the subjects after using the tool in the experiment.

External Validity: Due to number of subjects and the way they were selected, it can be said they are representative of the developers' population. The quantification of the factors was based uniquely in the experience of the subjects that had the opportunity to operate the tool in the time and the place they judged adequate. The interviewer helped the subjects to use the tool, but was not allowed to influence their evaluation.

Construct Validity: As much as the factors as the questions used in the study were based on taxonomies presented in the literature and, therefore, are considered valid for the study. A pilot experiment was conducted prior to the execution of the study to test the plan, the usefulness of the MACBETH tool, and to improve the study. The subjects were explained about the difference of the factors and characteristics of the questions pertaining to each factor. The possibility of guessing the result was eliminated by the use of paired comparison and the MACBETH tool, where the votes were checked for consistency. The percentile values for the risk factors were computed automatically, without the interference of the subject or the researcher. The software supported conflict resolution, by pointing the inconsistent votes to the subject, requiring their correction and thus leading to a better result.

Conclusion Validity: The T-test performed with a significant level of 0,05 (5%) leads to reliable conclusion about the adjustment values of risk factors obtained in the study. The same instruments were presented to the subjects and therefore, the implementation of the study can be considered reliable. The outlier elimination reduced the possibility of data misinterpretation and blocking the subjects and the projects in different groups minimized the heterogeneity of the elements. The adjustment values for risk factors are valid only for Information Systems projects and for the project sizes defined in this study, what turns it more restrictive, but more reliable.

Lessons Learned: The quantitative data obtained provide some positive indications of how the study was executed and about the trends of the adjustment values of each risk factor in software projects:

- It was observed the validity in the use of the MACBETH tool to accomplish the quantification of the factors. A qualitative questionnaire was presented to subjects to evaluate the methodology used during the study. The subjects informed that the MACBETH tool helped them in their decision process and could really express their personal opinion about the factors;
- Despite subjects' experience, it seems that the general concept that people have about the adjustment values

for risk factors is almost the same, regardless the project's size;

- According to the statistical analysis, some risk factors (*Analysis, Design, Coding, Testing, Planning, and Contracts*) can be considered having the same adjustment values regardless the size of the project;
- The remaining factors evaluated in the study (*Control, Team, Policies/Structure, and Clients*) must be represented by different adjustment values according to project size, but further investigation is still needed.

The values obtained from this study can be used as a guide for project managers during risk evaluation activities, providing quantitative data and showing a scenario of how the resources should be applied to contain certain risk types, what are the most important contingency plans, and where the risk management process requires more attention. Risk factors' adjustment values are used in the third step of the software project risk level evaluation technique presented in Section 2. For instance, regarding Table 2, to calculate the adjusted average values presented in the fourth row, we should use values from Table 6.

4. Related Works

The proposed approach of how to quantify the risk factors in software projects has some innovations when compared to previous works. Karolak [12] suggests a three level tree where the manager, at the first level, gives his opinion of how important are the 83 risks presented in a list that are divided in 10 different groups named factors. In a second level the averages of the values given previously are calculated and this number is attributed to the factor. Finally, at the third level, the probability of success of a project is calculated according to a weighted average of the tree categories that comprise the 10 factors. The managers, according to their own evaluation about the project, must determine the weights for these categories.

Other works can be found in the literature talking about the relevance of risk factors [13]. Nevertheless, they only show the opinion of specialists of what are the most important risks in a software project or what are their relations but do not quantify them nor determine their importance in the success of a project.

A valuable correlation that can be observed between this study and the ones presented in [13] is that the risk factors with the bigger adjustment values are the same indicated as the more import in that study.

5. Final Considerations

This paper presented an approach to calculate a project risk level based on its specific and systemic risks. An empirical study was planned and executed to determine the relevance of each risk factor to the success of a software project. Statistical treatment was given to the results to obtain the adjustment values needed by the methodology

and a questionnaire was created to help managers to evaluate the specific risks of a project.

The main contributions of this paper are to present a methodology to calculate the risk level as much of a factor as the whole project and also to quantify the adjustment value of each factor to help managers in their decision process.

Future perspectives of this work include the development of a tool to help managers in calculating a project risk level according to the methodology presented in this paper. The factor relevance weights can also be used to compare different projects and calculate a software organization's overall risk level based on the correlations observed among the factors.

The risk level obtained by applying the proposed approach can also be used to estimate the prices charged by a project based on its risk level. It can also be calculated for different project milestones to create a baseline during project planning and compare the risk levels throughout project execution, thus supporting the decision makers.

Finally, we intend to replicate the empirical study to enlarge the number of participants and verify if the results obtained in the first run can be generalized to other populations and other system categories.

Acknowledgements

The authors would like to thank all the subjects who took part in the empirical study for their valuable contribution, the organizations that allowed them to participate as well as the Brazilian Air Force, CAPES and CNPq for the financial support to this work.

References

- [1] Carr, M. J., Konda, S.L., Monarch, I., Ulrich, F.C., Walker, C.F., (1993), "Taxonomy-Based Risk Identification", Technical Report CMU/SEI-93-TR-6, Software Engineering Institute, Carnegie Mellon University, EUA, July
- [2] Jones, C. (1994) "Assessment and Control of Software Risks", Yourdon Press Computing Series. New Jersey
- [3] Karolak, D.W., (1996), "Software Engineering Risk Management", Los Alamitos, CA: IEEE, Computer Society Press
- [4] Barki H.; Rivard S.; Talbot, J., (1995), "Toward an Assessment of Software Development Risk". *Journal of Management Information Systems*, v. 10, n. 2, p. 203-225
- [5] Boehm, B.W. (1991), "Software Risk Management: Principles and Practices", *IEEE Software*, vol. 8, n. 1, January, pp. 32-41
- [6] Moynihan, T. (1997), "How Experienced Project Managers Assess Risk". *IEEE Software*, v. 14, n. 3, p. 35-41, May/June
- [7] Wallace, L., (1999), "The development of an instrument to measure software project risk". PhD Thesis - College of Business Administration, Georgia State University, Georgia.
- [8] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A., (2000), "Experimentation in Software Engineering – An Introduction", Kluwer Academic Publishers
- [9] Farias, L.L, Travassos, G.H., Rocha, A.R.C, (2003), "Managing Organizational Risk Knowledge", *Journal of Universal Computer Science*, Vol. 9, No. 7, pp. 670-681
- [10] Miranda, E., (1999), "Establishing Software Size Using the Paired Comparisons Method". Ericsson Research, Canada
- [11] Bana e Costa, C.A. & Vansnick, J.C., (1995), "A theoretical framework for Measuring Attractiveness by a Categorical Based Evaluation Technique (MACBETH)". In: Clímaco, J. *Multicriteria Analysis*. Berlin: Springer Verlag
- [12] Karolak, D.W., (1996), "Software Engineering Risk Management", Los Alamitos, CA: IEEE, Computer Society Press
- [13] Kleil, M., Cule, P.E., Lytnein, K., Schmidt, R.C., (1998), "A Framework for Identifying Software Project Risks". *ACM*, v. 41, n. 11, November

Software Traceability via Versioned Hypermedia*

Tien N. Nguyen, Ethan V. Munson, and Cheng Thao
Department of Computer Science
University of Wisconsin-Milwaukee

Abstract

Several researchers have explored the use of hypermedia technology to improve software traceability in software engineering tools. However, existing hypermedia-based tools have only limited supports for managing the evolutionary process of software traceability links among software documents. The Software Concordance (SC) improves software traceability maintenance and evolution by its versioned hypermedia traceability model. The paper presents the model and its implementations, which explicitly represent traceability links, allowing them to be browsed, visualized, and systematically analyzed. SC maintains and versions traceability link networks separately from documents, allowing developers to define multiple networks on the same set of documents, without modifying documents' contents. SC not only supports complex traceability linking structures (e.g., multi links), but also supports versioning of individual traceability links. Software documents and traceability link networks are all versioned at a fine granularity.

1 Introduction

Extensive effort in the software engineering community has been brought forth to improve the explicit connection of documentation and source code. The need for tools and techniques to maintain software traceability links in legacy systems is particularly important for a variety of software engineering tasks. These include general maintenance tasks, impact analysis, program comprehension, and more encompassing tasks such as reverse engineering for redevelopment and systematic reuse. However, the main limitation of existing software traceability tools is that traceability links are rarely explicit and their evolution over time are hardly recorded in a cohesive way as software projects evolve. During a software's life cycle, the traceability relationships actually connect *revisions* of software artifacts together. Let us take Figure 1a) as an example. Initially,

*This research was supported by the University of Wisconsin-Milwaukee Dissertation Fellowship.

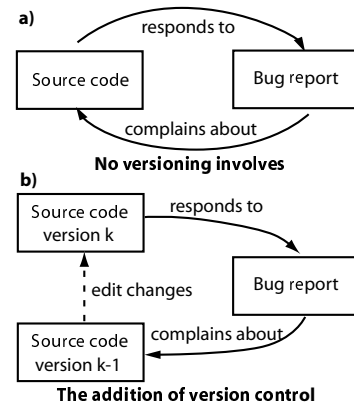


Figure 1. Traceability Links between Versions

an error is found in a source file and a bug report is created. To trace the error in the source file, a traceability link (“complains about”) is created, which points from the bug report back to the source file. Now, a developer fixes the error. For accountability, the developer adds another traceability link between two artifacts: the “responds to” link from the source file to the bug report. At this point, for his record, the developer does not remove the “complains about” link, therefore creating inconsistency. That is, the bug report would “complains about” the source file that has been fixed.

The problem is that the evolution of the networks of software artifacts and traceability links is not often recorded. When versions are added to the picture (see Figure 1b)), the bug report actually “complains about” the source file at version $k-1$, while the newer version of the source code is created in response to the bug report. That is, the traceability network between two documents at version $k-1$ contains the bug report, the link “complains about”, and the version $k-1$ of the source file. On the other hand, the version k of that traceability network consists of the version k of the source file, the “responds to” traceability links, and the bug report. Therefore, the whole process of software maintenance can be better recorded with the presence of version control.

Our solution to this problem is to provide a *versioned hypermedia* (also called *hypertext versioning*) traceability model and its infrastructures to manage versions of software artifacts and of traceability links. The traceability links are represented via a formal versioned hypermedia model that is easily supported by using underlying XML representations. The model addresses fine-grained version control of both documents and traceability links simultaneously. This is necessary because otherwise, linking would be limited to entire documents. The next section discusses related work on software traceability, followed by a description of our versioned hypermedia traceability model in Section 3. Section 4 describes its implementations in the Software Concordance (SC) environment. Last section is for conclusions.

2 Related work

Numerous techniques have been used for software traceability including cross referencing schemes [10], keyphrase dependencies [16], traceability matrices [8], hypertext [25], integration documents [17], formal models and languages [14], and constraint networks [4]. The ability to present interrelated information in non-linear form of the hypertexts has attracted many research in applying it to software traceability [28]. This section focuses only on *hypertext-based* software traceability approaches.

The visualization tools and systems using *HTML-based* hyperlinks, such as CHIME [9] and Javadoc [31], insert HTML tags into source code and make them browsable. Many tools produce Javadoc-based outputs for software documents based on program analyses [27], while Web-based tools take advantages of the WWW environment for supporting software engineering tasks [12]. SODOS [15] was based on a uniform document graph model and used a relational database to store a pre-defined set of relationship types. DIF [13] represented a hypertext network by storing relationships in a relational database. It supported traceability through keyword-based search and navigation mechanisms. Hypermedia-based integrated development environments (IDEs) that model a software system as a hypertext include DynamicDesign [3], ChyPro [1], SmallTalk literate programming [25], HyperDisco [35], component-based open hypermedia system [23]. Some traceability tools were focused on requirements and designs [30] and others on user cases and test cases [24]. The Chimera open hypermedia system [2] provides hypermedia services across multiple documents maintained by different applications. However, the versioning proposal for Chimera was not implemented [32]. TraceM [29] provides a framework for transforming implicit relationships to explicit ones. Many hypertext-based tools have been developed to improve consistency among software documents [5, 24].

Versioned hypermedia systems [33] offer an appealing

approach to representing the evolution of software documents and their traceability relationships. However, IDEs based on versioned hypermedia have only provided simple versioning of objects and never with versioning of links. Both RCS-based HyperWeb [11] and HyperCASE [7] did not support versioning for links. HyperPro [26] provided versioning for links via versioning for a composite, but no interactive program analysis was supported. DHT (Distributed Hypertext Systems) [22] was based on client-server architecture to provide integration of heterogeneous pre-existing software repositories and version control for software documents. Web-based software development can take advantage of WebDAV [34], an extension of HTTP to provide versioning for Web documents. Most of existing versioned hypermedia systems focused on hypermedia authoring and were not designed for software development.

3 Versioned hypermedia traceability model

3.1 Requirements

To address the evolutionary process of software traceability links, hypermedia in software engineering tools need to accommodate certain requirements. Firstly, the representation of hyperlinks must be explicit and facilitate systematic analyses of traceability networks. Implicit relationships hinders developers from having a full understanding of the system and from discovering important information. Secondly, hyperlinks must be able to connect many documents (i.e. multi links) and need to support both coarse-grained (at document level) and fine-grained linking (at documents' fragment level). Hypertext versioning should separate hypertext structures from documents' contents, giving developers more flexibility to have different traceability link networks to complete different tasks without modifying documents. In addition, the ability to track changes for a particular traceability relationship is very useful. The fact that existing hypermedia systems for software development do not version links is a significant factor preventing their wider use in the software engineering domain [33]. The history of traceability networks also needs to be recorded since it would help engineers to understand better the development of software documents and logical relationships among them over time. These requirements are addressed in our versioned hypermedia traceability model.

3.2 Documents and traceability links

Software documents and their traceability relationships can be modeled as a network of nodes and links where each node represents a fragment of a document and each link represents a relationship between fragments. To model software documents, SC follows a *structure-oriented* approach

where each document is considered to be logically structured into fine units, called *structural units*. Each software document is represented by a *document tree* in which each node encodes a logical unit of the document. This approach is often taken in *structured document* research, e.g. SGML and XML. Since XML has become the standard structured document format and very successful in representing many different data types, it is very natural to use XML for representing non-program artifacts. For a program, abstract syntax tree (AST) perfectly represents its logical structure. Via this approach, we have uniform structure-based representations for many types of software artifacts [20].

Because traceability is involved in the domain of linked documents, it is natural to use the *hypertext model* [6] as a basis for representing these relationships. Though different hypertext systems have variations of the notion, the hypertext model can be defined as a set of intellectual *works* and their inter- and intra-work relationships, represented by *links*. A *work* is an artifact that can be drawn from any medium, such as text, image, or video. A wide variety of terms have been used to describe a work in hypertext systems including *document*, *card*, *node*, *object*, and *component*. In the hypertext model, a *link* (or *hyperlink*) is a first-class entity and defined as an association among a set of works or anchors. Anchors denote regions of interest within a work and form the endpoints for links. The arity of a link specifies the number of its endpoints. While HTML supports only binary (two endpoint) links, more sophisticated representations permit *n*-ary links, which can have a variable numbers of endpoints.

We use this formal hypertext model as the basis for representing traceability relationships, and then extend it into a versioned hypermedia model via a fine-grained version control scheme. Each software document will be a *work* in the sense used by the hypertext model. It is possible to define an anchor corresponding to any well-defined structural unit. The SC's versioned hypermedia model is based on the following concepts: *linkbase*, *hypertext network*, *link*, and *anchor*. A linkbase is a container for hypertext networks and/or other linkbases. The relation between a linkbase and a hypertext network is the same as the relation between a directory and a file in a file system. A hypertext network, which represents for a traceability link network, can belong to only one linkbase. A hypertext network contains links and anchors. A link, representing for a traceability link, is *n*-ary and is an association among a set of anchors. An anchor can belong to multiple links. A link or an anchor can also belong to multiple hypertext networks. An anchor, denoting the region of interest within a document, refers to a structural unit. This separation between anchors and structural units allows for the separation between hypertext networks and documents' contents. This approach is called *hyperbase* [33] where the hypertext structures are

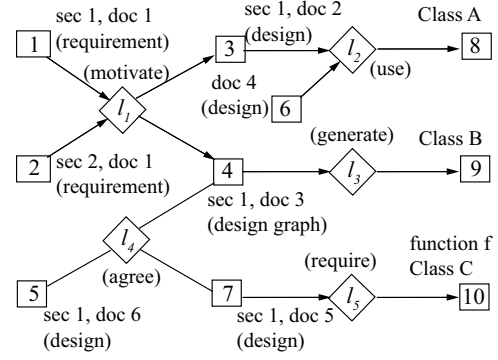


Figure 2. A traceability network

stored separately from documents. Links and anchors can be associated with any attribute-value pairs.

Links may have named types (such as “requires,” “must agree,” or “next item”), but the set of types is not fixed, so that different software processes can be supported. In SC, traceability links are divided into *causal* and *non-causal* classes. *Causal links* represent the relationships that carry with them an implied logical ordering of the documents involved. For example, testing and bug reports cannot be produced until an implementation is available, and while it is not necessarily the case that requirements will be written before designs, there is certainly a logical relationship between them that makes design depend on requirements. A causal relationship can be considered as a *relation* between entities: something happens and causes something else to happen. Causal links are always directional, connecting a set of source anchors and a set of target anchors. *Non-causal links* exist when documents or parts of them must agree with each other, but the causality cannot be clearly identified. Figure 2 shows an example of a traceability network. The links l_1 , l_2 , l_3 , l_5 are causal links, which have incoming and outgoing edges that are directed, while l_4 (non-causal) and its anchors are connected by non-directed edges. The anchors (1-10) refer to document nodes.

3.3 Fine-grained versioning scheme

To provide version control for software documents and traceability link networks, it is obvious that a versioning framework for trees and directed graphs is required. This section describes our fine-grained versioning scheme for the trees and directed graphs. The versioning scheme is based on nodes and attributes in an attribute table. A tree is defined with two main attributes: 1) the “children” attribute maps each node to a sequence holding its children, and 2) the “parent” attribute maps each node to its parent. Figure 3 illustrates this via an example. In the example, a “content” attribute is also defined to hold a string value for some of the

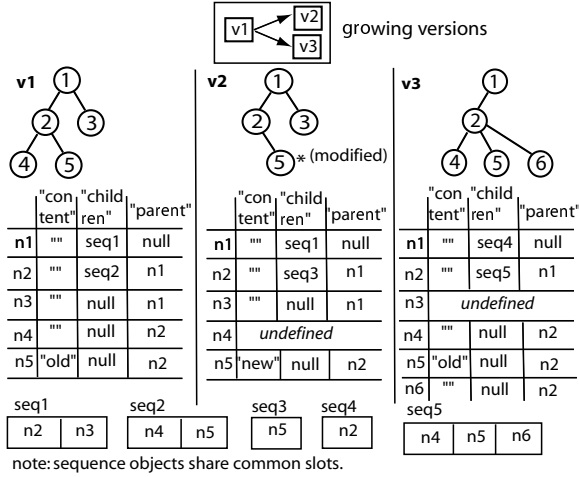


Figure 3. Versioning for document trees

nodes. Assume that there are three versions: $v1$, $v2$, and $v3$. Versions $v2$ and $v3$ branch off from version $v1$. The shape of the tree at each of the three versions is shown. Version $v2$ has two differences from the version $v1$: node 4 was deleted and the content of node 5 was changed. Version $v3$ has an inserted node (node 6) and node 3 was deleted. The values of cells in the attribute table changed to reflect modifications to the tree in these versions. For example, at the version $v2$, the “content” cell of node 5 contains a new value (the string “new”), and the “children” cell of node 2 contains a reference to a new sequence object ($seq3$). $Seq3$ contains a reference to node 5 since node 4 has been deleted. If there is a request for the values of cells associated with node 4 at $v2$, a run-time error will be reported.

Versioning for a directed graph is similar except that the attribute table does not have the “parent” attribute for nodes. Figure 4 shows an example of a traceability network. Figure 4a) and Figure 4d) display the network at two versions $v1$ and $v2$. The directed graphs representing for the network’s structures at two versions are in Figure 4b) and Figure 4e). Links’ nodes (e.g. nodes 2 and 6) have edges coming into them and do not refer to anything. Figure 4c) shows part of the attribute table for the network at version $v1$. The “ref” cell for an anchor node (e.g. node 1) contains a reference to the corresponding document node (e.g. $n(section1)$). Figure 4f) shows the attribute table at version $v2$. Node 5 is deleted. Node 3 now has only one child. Node 9 (representing $link3$) and node 10 (representing $class3$) are created. The “ref” cell for node 10 points to $class3$.

3.4 Configuration management

Versioning for documents and for traceability networks is accomplished via the fine-grained versioning scheme. The issue of how a version of a traceability network can

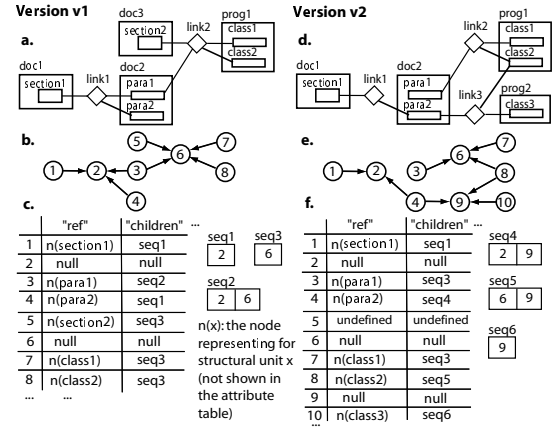


Figure 4. Versioning for traceability networks

have its anchor nodes pointing to the proper versions of document nodes is addressed by our product versioning configuration management, *Molhado* [21]. *Molhado* is built as part of our research. Instead of focusing on individual documents, *Molhado* versions a software project as a whole. All objects including documents and traceability networks are versioned in a *uniform, global version space*. In *Molhado*, a *version* is global across the whole project and is a point in a *tree-structured discrete time* abstraction, rather than being a *particular state* of an object as in classical versioning systems (e.g. CVS [19]). The state of the whole software system is captured at certain discrete time points and only these captured versions can be retrieved in later sessions.

The *current version* is the version designating the current state of the project and it is *global* across the project. Note that the attribute tables can be seen as having the third dimension: the version. Depending on current version, the cells’ values might be different. When the current version is set to one of captured versions, the state of the whole project and attribute tables are set back to that version. Therefore, document trees and traceability networks will get the proper shapes and anchor nodes in the traceability graph will also correctly refer to document nodes at the current version. Initially, users explicitly specify the current (working) version. Then, the graphical user interfaces automatically detect the current version based on the current editing window. Any changes made to the project at the current version create a temporary version, *branching off* from the current version. That temporary version would not be recorded if users did not require. Users primarily operate at the project level. To record the history of an individual document, they capture the whole project at a version. Capturing the whole project is quite efficient because the versioning system only records changes and works at a very small granularity. Details on *Molhado* can be found in another document [21].

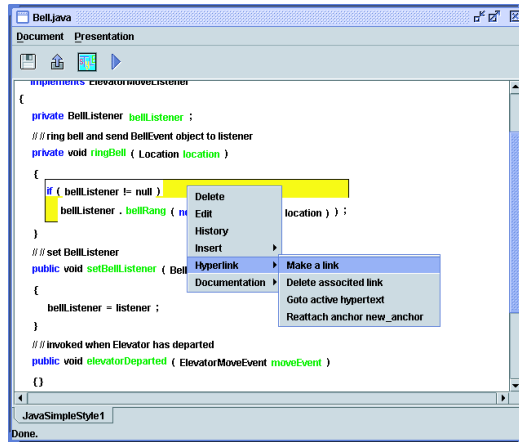


Figure 5. Java structured editor

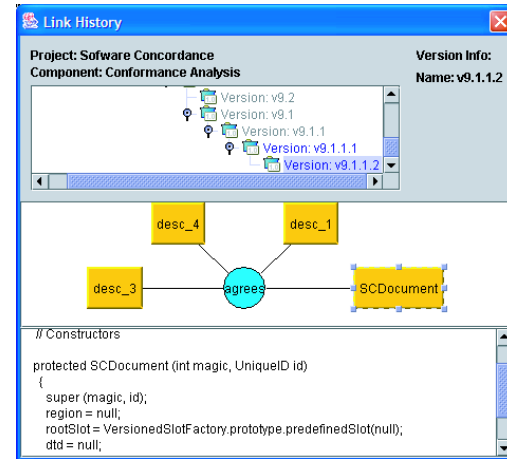


Figure 6. Versioning for a link

4 Implementations

The versioned hypermedia traceability model has been implemented and integrated into the Software Concordance (SC) IDE [20]. Versioned hypermedia functionality in SC helps developers to manage versions of software documents including Java programs, multimedia documentation in XML (multimedia file are stored separately and not versioned), HTML, ASCII text, Scalable Vector Graphics (SVG) documents, and UML diagrams. The system is able to import and export its internal binary documents from and to external formats such as XML, HTML, SVG, and ASCII text at any version. When a user choose to edit a document, appropriate built-in editors will be invoked. The editors are all hypertext-savvy and version-savvy. Figure 5 shows the Java structured editor. When the user right-clicks on a document node, a popup menu is displayed to allow the user to view the version history of the document node, to create an anchor at the node and add it to a traceability network, etc. From a document node, the user is able to navigate among documents via the traceability networks. SC also supports embedded HTML links within Java programs and they do not interfere with program analyses [20].

Traceability link services allow for the manipulation and versioning of linkbases and traceability link networks. The user can create or delete a linkbase or a network, restructure linkbases, open an existing network, import and export a network from and to XLink [36] format at any version. The user can also view the history of a network or of an individual traceability link. Figure 6 shows the history of a network containing the link “agrees”. Note that the link was not created until the version v9.1.1.1. Therefore, the earlier versions on the top window are “disabled”. The class “SCDocument” was displayed in the bottom window since the user clicked on the corresponding anchor.

From the popup menu in a document editor, the user is

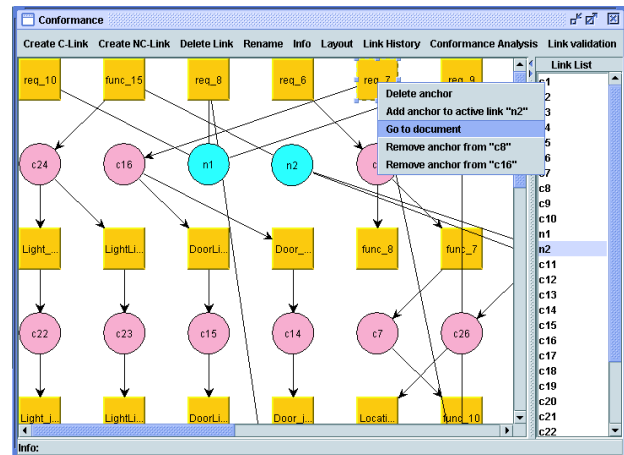


Figure 7. A hypertext network editing window

able to open any traceability network related to the current document node (see Figure 7). A circle represents for a link, a rectangle for an anchor. Services for links include link creation, deletion, renaming, attribute’s value viewing, and link history viewing. Anchor services include deleting an anchor, adding an anchor into the active link, removing an anchor off some link, renaming an anchor, and opening the structural unit that the anchor refers to. When the user is ready to record the state of the project after modifying networks or documents, a capture command can be issued and a new version is created. The user does not need to check in or check out documents or hypertext entities individually.

5 Conclusions

The SC’s versioned hypermedia traceability model allows for the management of the evolution of software arti-

facts and traceability links simultaneously in a *fine-grained* manner. SC not only manages the versions of traceability link networks but also supports for versioning of individual links. Traceability links are explicitly represented, visualized, and facilitate systematic software traceability analyses. The SC's versioned hypermedia infrastructures provide the foundation for our current research on how to automatically infer traceability links using Latent Semantic Indexing technique [18]. The preliminary results from our experimental studies show that the performance and time efficiency of the SC system is satisfactory.

References

- [1] M. Amsellem. ChyPro: A hypermedia programming environment for SmallTalk-80. In *Proceedings of ECOOP*, 1995.
- [2] K. M. Anderson, R. N. Taylor, and E. J. Whitehead, Jr. Chimera: hypermedia for heterogeneous software development environments. *ACM Transactions on Information Systems (TOIS)*, 18(3):211–245, 2000.
- [3] J. Bigelow and V. Riley. Manipulating source code in DynamicDesign. In *Proceedings of Hypertext Conf.*, 1987.
- [4] J. Bowen, P. O'Grady, and L. Smith. A constraint programming language got life-cycle engineering. *AI in Engineering*, 5(4):206–220, 1990.
- [5] S. Choi and W. Scacchi. Formalization and tools supporting structural correctness of software life cycle descriptions. In *Proceedings of IASTED Conf. on Soft. Engineering*, 1998.
- [6] J. Conklin. Hypertext: An introduction and survey. *IEEE Computer*, 20(9):17–41, 1987.
- [7] Cybulski and Reed. A Hypertext Based Software Engineering Environment. *IEEE Software*, 9(2):62–68, March 1992.
- [8] A. M. Davis. *Software Requirements: Analysis and Specification*. Prentice Hall, 1990.
- [9] P. Devanbu, Y.-F. Chen, E. Gansner, H. Müller, and J. Martin. CHIME: customizable hyperlink insertion and maintenance engine for software engineering environments. In *Proceedings of the 1999 International Conference on Software Engineering*, pages 473–482, 1999.
- [10] M. Evans. *The Software Factory*. John Wiley, 1989.
- [11] J. C. Ferrans, D. W. Hurst, M. A. Sennett, B. M. Covnot, W. Ji, P. Kajka, and W. Ouyang. HyperWeb: a framework for hypermedia-based environments. In *Proceedings of the Symposium on Software Development Environments*, pages 1–10. ACM Press, 1992.
- [12] R. Fielding, J. Whitehead, K. Anderson, P. Oreizy, G. Bolcer, and R. Taylor. Web-based development of complex information products. *Communications of the ACM*, 41(8):84–92, August 1998.
- [13] P. K. Garg and W. Scacchi. A hypertext system to manage software documents. *IEEE Software*, 7(3):90–98, May 1990.
- [14] V. Hamilton and M. Beeby. Issues of traceability in integrating tools. In *Proceedings of the IEEE Colloquium, Computing and Control Division*. IEE, 1991.
- [15] E. Horowitz and R. Williamson. SODOS: a software documentation support environment—its use. *IEEE Transactions on Software Engineering*, SE-12(11):1076–1087, Nov 1986.
- [16] J. Jackson. A keyphrase based traceability scheme. In *Proceedings of the IEEE Colloquium, Computing and Control Division*. IEE, 1991.
- [17] M. Lefering. An incremental Integration Tool Between Requirements Engineering and Programming in the Large. In *Proceedings of the IEEE Symposium on Requirement Engineering*. IEEE, 1993.
- [18] J. I. Maletic, E. V. Munson, A. Marcus, and T. N. Nguyen. Using a hypertext model for traceability link conformance analysis. In *Proceedings of the Automated Software Engineering (ASE), Traceability Workshop*, 2003.
- [19] T. Morse. CVS. *Linux Journal*, 1996(21es):3, 1996.
- [20] T. N. Nguyen and E. V. Munson. The Software Concordance: A New Software Document Management Environment. In *Proceedings of the 21th International Conference on Computer Documentation*. ACM Press, 2003.
- [21] T. N. Nguyen, E. V. Munson, and C. Thao. Structured software configuration management for web projects. In *International World Wide Web Conference*. ACM Press, 2004.
- [22] J. Noll and W. Scacchi. Supporting software development in virtual enterprises. *Journal of Digital Information*, 1(4), February 1999.
- [23] P. Nurnberg. Extensibility in component-based open hypermedia systems. *Journal of Network and Computer Applications*, 4(1), 2001.
- [24] T. Olsson and J. Grundy. Supporting traceability and inconsistency management between software artifacts. In *Proceedings of the 6th IASTED International Conference Software Engineering and Applications (SEA 2002)*, 2002.
- [25] K. Østerbye. Literate SmallTalk using hypertext. *IEEE Trans. on Soft. Engineering*, 21(2):138–145, Feb 1995.
- [26] K. Østerbye and K. Nørmark. An interaction engine for rich hypertext. In *Proceedings of the 1994 ACM European conference on Hypermedia technology*, pages 167–176, 1994.
- [27] J. Sametinger and M. Riebisch. Evolution support by homogeneously documenting patterns, aspects, and traces. In *Proceedings of the CSMR Conference*, 2002.
- [28] W. Scacchi. Hypertext for software engineering. *Encyclopedia of Software Engineering*, 2001.
- [29] S. Sherba, K. Anderson, and M. Faisal. A framework for mapping traceability relationships. In *Proceedings of the ASE 2003 Traceability Workshop*, 2003.
- [30] G. Spanoudakis. Plausible and adaptive requirement traceability structures. In *Proceedings of the Conference on Software Engineering and Knowledge Engineering*, 2002.
- [31] Javadoc tool home page. <http://java.sun.com/j2se/javadoc/>.
- [32] E. J. Whitehead, Jr. A proposal for versioning support for the Chimera system. In *Proceedings of the Workshop on Versioning in Hypertext Systems*. ACM Press, 1994.
- [33] E. J. Whitehead, Jr. *An Analysis of the Hypertext Versioning Domain*. PhD thesis, University of California – Irvine, 2000.
- [34] E. J. Whitehead, Jr. WebDAV and DeltaV: collaborative authoring, versioning, and configuration management for the Web. In *Proceedings of the Hypertext Conf.*, 2001.
- [35] U. K. Wiil and J. J. Leggett. The HyperDisco approach to open hypermedia systems. In *Proceedings of the ACM conference on Hypertext*, pages 140–148. ACM Press, 1996.
- [36] W3C XML Linking. <http://www.w3c.org/XML/Linking>.

Specification and Validation of Transactional Business Software: An Approach Based on the Exploration of Concrete Scenarios

Alexandre Correa, Cláudia Werner
COPPE/UFRJ – Federal University of Rio de Janeiro - Brazil
{alexcorr, werner}@cos.ufrj.br

Abstract. This paper presents an approach to the specification and validation of transactional business software. The focus of this work is on the production of detailed use case specifications and on the precise definition of all transactions and business rules using a subset of UML class diagrams and statecharts combined with textual specifications written in OCL (Object Constraint Language). We show how to produce and validate such artifacts using a scenario driven approach combined with animation and prototyping techniques in a highly iterative process. The paper also presents PSW (Precise Specification Workbench), a tool that supports the proposed approach.

1. Introduction

One of the goals of the requirements activities in software engineering is to produce a clear, consistent, precise and unambiguous specification of the system that is to be developed. Among the scenario-based techniques that have been proposed by the requirements engineering community to understand, model and validate software requirements [15], use case based specification [2] is one of the most widely used, particularly in object oriented development environments. Use cases are very popular because of their informal, easy to use style, which caters to technical as well as non-technical stakeholders of the software under development. However, the large variance of word meanings in natural language has always posed problems for those who attempt to construct an unambiguous and consistent specification, as extensively described in the technical literature [4] [11].

Our research is focused on the elaboration and validation of precise specifications for transactional business software, particularly in the financial domain. The requirements specification of transactional business software usually concentrates on its functional and structural (data) essentials, whereas life-cycle aspects are often restricted to a few model elements. Moreover, such systems must deal with many complex business rules that are often not fully and correctly uncovered when an

informal approach to software specification is applied. This is of particular importance to the financial domain, where a requirement error or misunderstanding can result in a significant loss of money and credibility.

There are a number of approaches such as the Rational Unified Process [10] that tries to specify the functional aspects of a software system combining natural-language use case descriptions with object-oriented models and, nowadays, the Unified Modeling Language (UML) [13] is regarded as the de-facto standard for the elaboration of such models. By reviewing the specifications produced by graduate students and software developers from the industry using such approaches in more than 30 transactional business software projects, we have noticed some recurrent problems and two of them are particularly relevant to this work. First, system analysts often focus too prematurely on use case interaction details, leaving the precise definition of transaction results and business rules to a later stage or, even worse, only loosely defining them. This generates omissions, inconsistencies and significant later rework on the specification.

A second recurrent problem observed is the absence of automated support for the exploration, organization and reuse of concrete scenarios. Concrete scenarios force us to address the “devil in the detail” during requirements specification and validation, and reasoning with concrete examples as well as abstract models helps on comprehension by building a memory schema that link the specific (scenario) to the general (model) [15]. However, currently available object oriented CASE tools focus on the production of abstract models and specifications, and, as a result, concrete scenario exploration has to be manually done or it is not done at all.

This paper presents an approach and tool support to the specification and validation of transactional business software that combines formal and informal techniques, aiming at alleviating the problems just described. In the proposed approach, before producing a detailed use case specification, the underlying transactions and business rules are defined by means of UML class diagrams and statecharts combined with textual specifications in OCL

(Object Constraint Language) [12]. We use a scenario driven approach combined with animation and prototyping techniques in a highly iterative process, which allows us to reach a precise definition in small and validated steps. After precisely understanding the transactions involved in a use case, we address the interaction issues and the associated non-functional requirements, elaborating a use case textual specification consistent with the UML structural and behavioural model.

This paper is structured as follows: Section 2 details our approach to the production of precise specifications. Section 3 presents an overview of PSW (Precise Specification Workbench), a tool that supports the proposed activities. In Section 4, related works are discussed and the conclusions are drawn in Section 5.

2. Proposed Approach

The techniques and tool support described in this paper are focused on the production of a detailed use case specification, which is the result of late-phase requirements activities. We assume that the software development team has already a reasonably good understanding of the organizational environment and stakeholders needs, and that an initial use case model has been produced as a result of early-phase requirements activities. This initial use case model should contain a preliminary description of the main use cases along with a summary section that describes the use case goal and a brief outline of the sequence of underlying transactions.

The proposed approach consists of four main stages. The first stage corresponds to the definition of the main results expected from each use case transaction. The second stage uncovers all computation and inference rules behind the production of those results. The third stage explores all rules that could deny or restrict the transaction execution. Finally, the fourth stage focus on the use case textual specification detailing all interactions between the actors and the system that will trigger the use case transactions. At this final stage, we also consider non-functional aspects and their influence on the use case interactions.

2.1. Specification of the Main Expected Results

At the first stage, our goal is to define the main results expected from each transaction contained in a use case. To reach this goal, we explore all relevant scenarios associated to each transaction, one at a time. For each scenario, we begin by informally stating the expected results in natural language. Figure 1 shows an example for the *Rent Copies* use case of a video rental system. Having reached an initial agreement on the informally defined results, we evolve an UML conceptual model that should

contain all information necessary to the formalization of the transaction results. Figure 2 shows an excerpt from the conceptual model of the video rental example.

Use Case: Rent Copies
Transaction: Rent Copies
Abstract Scenario: Client asks for the rental of copies that he has picked up from the shelf.
Input:
• Client and Copies to be rented
Expected Results:
Record of the rental and a Receipt containing:
• Client who have rented the copies (code, name and address)
• Copies rented (code, film title, expected date of return, rental fee)
• Rental Total Fee, Discount, Rental Net Fee, Rental Date

Figure 1- Abstract scenario

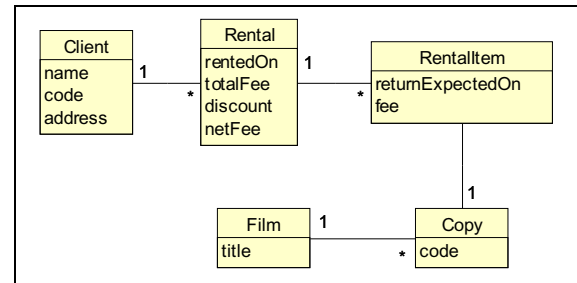


Figure 2- Video Rental Conceptual Model

Next, we restate the expected results of the transaction by exploring concrete scenarios, i.e., instantiations of an abstract scenario. In the video rental example, a concrete scenario could be “*Client John Miller asks for the rental of a copy of the Gladiator film that he has picked up from the shelf*”. For each concrete scenario, we define the system state just before and after the transaction execution by instantiating elements defined in the conceptual model. At this stage, this definition focuses only on the instances and links that are created, modified or deleted as a result of executing the transaction. That is what we call the main results of a transaction. We leave to a later stage details about the exact state of each instance created or modified. The scenario exploration described above is fully supported by PSW (Precise Specification Workbench), a tool developed in the Software Engineering Laboratory at Federal University of Rio de Janeiro to provide automated support to the proposed approach (section 3 presents more details about PSW). During the exploration of these concrete scenarios some modifications in the conceptual model may be needed as a result of a deeper understanding of the concepts involved in these scenarios.

Having reached a consensus on the expected results of the transaction, we formally state those results in OCL. Each transaction is captured as an operation of a type named *System* that embodies all the types defined in the conceptual model. Thus, *System* represents a black box

view of the software, and its operations represent the transactions that will be triggered inside the use cases. The expected results are expressed as OCL post conditions. Figure 3 illustrates a first version of the postconditions for the Rent Copies transaction.

```
context System::rentCopies
(aClient : Client, copiesRented : Set(Copy))
post:
-- newRentals is the set of Rentals associated to aClient that
-- were created as a result of the execution of System::rentCopies
let newRentals : Set(Rental) = aClient.rental->select(aRental |
aRental.ocllsNew()),
-- newRental is the first element of newRentals.
newRental : Rental = newRentals->asSequence()->first()
in
-- only one new Rental was created
newRentals->size() = 1 and
-- for each rented copy, one new instance of RentalItem was
-- created and associated to the rented copy and to newRental.
copiesRented->forall(copy | newRental.rentalItem->
one (item | item.ocllsNew() and item.copy = copy))
post:
-- all rented copies are in the state Rented (as defined by the
-- statechart associated to the Copy class)
copiesRented->forall(copy | copy.ocllnState(Rented))
```

Figure 3 – RentCopies post conditions in OCL

At the last step of this stage, we check if the OCL specification correctly captures the transaction semantics using the animation features of PSW. The concrete scenarios explored up to this point are used to validate the OCL transaction specification. More details on the animation features of PSW are described in section 3.

2.2. Computation and Inference Rules

At the second stage, our goal is to refine the specification of the transaction results by identifying and specifying all the rules involved in the production of the results. Computation, inferred knowledge and action enabler rules should be discovered and precisely specified.

First, we identify all decisions and knowledge that are necessary to produce the expected results. Then we identify all rules necessary to make a decision or to generate a specific result. Each rule is first defined in natural language and then formalized in the conceptual model. Figure 4 shows some additions (*Category* type and *rentalFee* operation) made to the conceptual model in order to express the rules related to the computation of the rental fee of a film copy.

Each rule definition is validated through the animation of concrete scenarios using the PSW's UML/OCL animator module. These scenarios are defined by using conventional testing techniques. First, an object space is defined by instantiating the relevant types defined in the conceptual model. Figure 5 shows an example of an object space that would be used in the animation of the

rentalFee rule. Then, one or more OCL expressions and their expected results are defined and submitted to the animator. The animator evaluates the expressions using the given object space and displays the results. Figure 6 shows some concrete scenarios that could be used in the validation of the *rentalFee* rule.

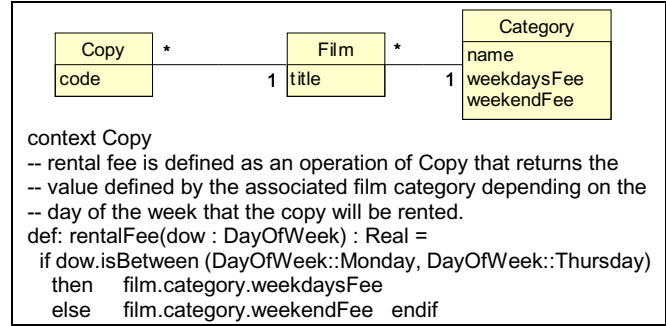


Figure 4 – Rule formalization

Copy			
Alias	Code	State	Film
Copy1	112233	On the shelf	Film1
Film			
Alias	Title	Category	
Film1	Gladiator	Category1	
Category			
Alias	Name	WeekdaysFee	WeekendFee
Category1	Special	4.00	6.00
Category2	Normal	3.00	5.00

Figure 5 – Object space for rentalFee rule animation

Expression	Expected	Actual
Copy1.rentalFee (DayOfWeek::Monday)	4.00	
Copy1.rentalFee (DayOfWeek::Friday)	6.00	

Figure 6 – Scenarios for rentalFee rule evaluation

2.3. Constraints and Preconditions

After capturing the rules necessary to produce the results of a transaction, we refine the transaction specification by identifying and specifying constraints and preconditions. We should investigate circumstances within a transaction that either are not acceptable or would deny its execution.

Decision: When the client is forbidden to rent copies?

Rule 1: IF a client has one or more overdue copies THEN he is forbidden to rent copies.

Rule 2: IF a client has a balance less or equal than the debit limit THEN he is forbidden to rent copies.

Rule 3: IF a copy is not returned on the expected day of return THEN it is overdue.

Figure 7 – Constraints: rentCopies transaction

The rules are first expressed in natural language. Figure 7 shows an example for the rentCopies transaction.

Next, they are formalized in the conceptual model. As illustrated in Figure 8 and Figure 9, rules 1, 2 and 3 are expressed as preconditions of the *rentCopies* transaction and as new properties defined in the *Client* type, as well.

```
context Client
-- overdueCopies is an operation of Client that returns the number
-- of rentalItems (from all rentals associated to a client) that have
-- not been returned yet.
def: overdueCopies() : Integer =
self.rental.rentalItem->select(i | i.oclnInState(NotReturned)->size())
```

Figure 8 – Rule formalized as a property of Client type

```
context System::rentCopies
(aClient : Client, copiesRented : Set(Copy),
reservations : Set(Reservation))
-- preconditions for the rentCopies transaction:
pre noOverdueCopies : aClient.overdueCopies() = 0
pre balanceAboveLimit : aClient.balance() > aClient.debitLimit()
```

Figure 9 – RentCopies pre conditions in OCL

The final step of this stage is the validation of the refined version of the transaction specification and all rules involved. Again, we use the animation features of PSW to walk through concrete scenarios as described in the previous sections.

2.4. Detailed Use Case Specification

Having fully specified the transactions involved in a use case, we produce a refined version of the use case specification in natural language. Now we shift focus to all the interactions that are necessary to trigger the execution of the underlying transactions considering usability and other non-functional aspects.

In the Rent Copies use case, for example, we must define how the user will interact with the system in order to trigger the *rentCopies* transaction, which has as input a client, the copies to be rented and a set of reservations eventually made by the client. We need to make some decisions as, for example, how the user will input the client or what will be the interactions and in which order they will be carried out in order to trigger the transaction. There are many alternatives to this: each client has a card and the clerk scans this card, the clerk may enter the client code, the clerk may do a search based on the client name, or we can assume that the client is already known (precondition), probably as the result of the execution of another use case. At this stage we should use available guidelines to the organization of a written use case specification, such as the ones described in [2]. The resulting use case specification may be validated by user interface prototypes that can be integrated to the PSW's animation features through an API, as described in the next section.

3. Tool Support

This section briefly describes PSW (Precise Specification Workbench), a tool designed to support the approach presented in this paper. PSW functionalities allow clients and developers to explore concrete scenarios of transactions and rules of the software to be developed

PSW was designed as an add-on to existent UML CASE tools. Therefore, all diagrams of the conceptual model may be elaborated in any external UML CASE tool featuring a XMI export capability, while the OCL specifications are produced in textual files and analyzed by PSW OCL compiler, which is fully compliant with OCL 2 specification. The most important modules of PSW are described in the following paragraphs.

The object space manager module allows the creation, retrieval and storage of different object space configurations. An object space configuration is a set of instances and links between instances of types defined in the conceptual model that can be created through the invocation of some basic operations: *create an instance*; *delete an instance*; *create a link between two or more instances*; *remove a link*; *modify the value of one or more attributes of an instance*. Each object space has a name and can be used as the initial or final system state of one or many scenarios. As the user creates or modifies an object space configuration, this module indicates all invariants that may have been violated. The user can also selectively turn on and off the invariant check feature for some or all invariants of the model, allowing him to concentrate only on the elements required to explore a specific scenario.

The ad-hoc OCL expression evaluator module evaluates any OCL query expression against an object space configuration. The analyst can use this feature as an aid to build the specification of a complex rule or transaction by interactively evaluating its parts. This module can also be used to inspect the state of any element of the system during the exploration of a scenario.

The scenario manager module allows the creation and management of all scenarios necessary to produce and validate the OCL specification. A scenario may be associated to a rule, to a transaction or to a use case. Each scenario has an informal description in natural language, an initial object space configuration, that may be one of the configurations already stored in PSW or a new one, the sequence of rules or transactions that will be executed and, optionally, the expected results.

The UML/OCL animator module supports the validation steps of the approach through animation of the specification. The animation of a computation or inferred knowledge rule is done through the simple evaluation of its associated scenarios. A scenario for this kind of rule is defined by an object space configuration, an OCL

expression corresponding to the invocation of a query operation specifying the rule to be evaluated and the expected result. Figure 6 shows two scenarios associated to the *rentalFee* rule. The animator evaluates all the expressions of the scenarios associated to the rules that the user has selected for animation and displays the results using a green/red background, visually indicating the scenarios where the evaluation matched or not the expected results.

The animation of rules and transactions that are not expressed as query operations is a bit more complex since it cannot be done by simply evaluating one or more OCL expressions. PSW offers two forms of animation of such operations. The first one is a simple post condition evaluation and it may be performed in interactive or batch modes. In the interactive mode, the user defines an initial object space configuration and indicates to PSW the operation that will be animated. PSW checks the object space against all preconditions specified for the operation. The user invokes any of the commands available in the object space manager to modify the initial object space configuration, thus animating the effects produced by the execution of the operation. After all desired effects have been submitted to the animator, the user signals the end of execution. Then, PSW checks all invariants and post conditions specified for the operation reporting any violation that eventually exists in the resulting object space. In the batch mode, the user selects one or more scenarios to be animated. Each scenario consists of an initial object space configuration, an operation call and the resulting object space configuration. Then, PSW checks if the initial and final object space configurations violate any of the invariants, pre and postconditions defined for the operation, reporting the scenarios where violations have occurred.

The second form of animation is the filmstrip commands generation. Instead of only checking the post conditions in an object space manually configured by the user, PSW can generate the commands given an operation specification and an initial object space configuration. This form of animation may also be done in interactive or batch modes. In the interactive mode, the user defines an initial object space configuration and indicates to PSW the operation that will be animated. PSW checks the object space against all preconditions specified for the operation and searches for the minimal set of commands that generates an object space configuration satisfying the invariants and post conditions of the operation. The complexity of the search process is controlled by a set of heuristics that avoids a search space explosion problem. For deterministic specifications, PSW generates the resulting object space configuration, and the user may use PSW's OCL query capabilities to see the results or ask PSW to list all generated commands. For non-deterministic specifications, the user may ask PSW to

generate only one result configuration and interactively ask for other configurations until all configurations have been generated. In the batch mode, the user selects one or more scenarios to be animated. Each scenario consists of an initial object space configuration, an operation call and a resulting object space configuration. For each scenario, PSW generates the commands as detailed before and tries to match the resulting object space against the expected object space as specified by the scenario. This is done until a match is found or until no other configurations can be produced. This feature is very useful to detect under or over constrained post conditions specifications.

The batch mode of animation may also be used to perform a "regression animation" of the scenarios. Selected scenarios are animated and PSW reports all deviations from the expected results. This feature is very useful in an iterative development approach.

In order to further involve the client in the validation process, it is possible to integrate the animation facilities of PSW with user interface prototypes. This integration is done through PSW-API that makes all functionality provided by the object space management, OCL expression evaluator and the UML/OCL animator modules available to a programmatic use. Since all functions available in the API are accessible through web services, a wide range of languages may be used in the user interface implementation. The user interface prototype enables the stakeholders to "execute" the specification by entering data, triggering the execution of transactions and observing the resulting behaviour.

4. Related Work

Most of the recent component based development methods ([1], [3]) emphasize the importance of a precise specification and adopt OCL as the specification language. However, those methods do not clearly connect the elaboration of UML/OCL models with the production of use case specifications, nor emphasize the exploration of concrete scenarios. One contribution of our work is to provide a practical but systematic approach to the elaboration of use cases for transactional business software that is based on a step-by-step exploration of scenarios through the animation of a precise conceptual model and all the underlying business rules.

PSW's UML/OCL animator module is closely related to the animation of classical formal languages. There are several animation tools that automatically execute or interpret formal specifications produced with languages such as Z [8] and VDM-SL [5]. Some of them can be integrated with UML class diagrams, but the user must also know the underlying formal language to interpret the results.

Another contribution of our work is the tool support to the elaboration and validation of UML/OCL

specifications using a scenario-driven approach. Nowadays, tool support for OCL development is very scarce, and this is an important factor that limits the widespread adoption of OCL by industry. The USE (UML-based Specification Environment) [14] is a tool that supports indirect animation of UML/OCL design models by means of a script language. This tool is oriented to validating detailed design models, while PSW is oriented to the validation of more abstract models. USE animation feature is very much similar to the simple post condition evaluation form of animation described in section 3. The second form of animation offered by PSW is not supported by USE. Moreover, USE does not offer a direct integration with other CASE tools (the conceptual model is defined in a proprietary language) and does not have the object space manager, scenario management and the web services API modules featured by PSW.

5. Conclusion

We have presented an approach and tool support to the specification and validation of transactional business software. The main point of our approach is the precise definition of all transactions involved in a use case, supported by scenario-based validation through the use of animation and prototyping techniques, as a means to produce a more solid use case detailed specification.

Though specification animation and prototyping are not new, we provide significant contributions to the UML/OCL community by supporting the use of techniques and tools present in formal environments and making them available to a wider public. We are aware that animation, like other testing oriented techniques, can never prove that a model is consistent, correct or complete [9]. However, our experience, like others described in the technical literature ([6] [7]), has shown that by bringing a specification to life, animation enables clients and developers to provide valuable feedback and to detect problems earlier in the development life cycle. When combined with other techniques such as inspections, for example, it provides a solid framework for the validation of specifications.

We have used the proposed approach in one industrial project in the financial domain. In the beginning, a conventional RUP approach was being followed and we noticed that the analysts were bouncing between actor-system interaction, domain, transaction and business rules issues without adequate guidance for producing the specification. As the team started to use the approach presented in this paper, most of the already specified use cases demanded modifications due to errors, inconsistencies and misunderstandings. Moreover, the exploration of concrete scenarios supported by PSW has established a very powerful means of communication between developers and customers. Although initial

results have shown a reduction on requirements defects, increased precision on communication and less rework on use case specifications, the cost-effectiveness of our approach is a particular aspect that needs to be addressed to make it attractive to a wider range of industrial projects. Therefore, as future work, we plan to conduct formal experiments to further investigate this issue.

References

- [1] J. Cheesman, J. Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*, Addison Wesley, 2001.
- [2] A. Cockburn, *Writing Effective Use Cases*, Addison-Wesley, 2000.
- [3] D. D'Souza, A.C. Wills, *Objects, Components and Frameworks with UML. The Catalysis Approach*, Addison-Wesley, 1998.
- [4] S.M. Easterbrook, J. Callahan, "Formal Methods for V&V of Partial Specifications: An Experience Report", *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, Maryland, USA, 1997.
- [5] R. Elmstrom, P.G. Larsen, P.B. Lassen, "The IFAD VDM-SL Toolbox: A practical approach to formal specifications" *ACM SIGPLAN Notices* 29(9), 1994.
- [6] P.Fenkan, H.Gall, M.Jazayeri, "Visual Requirements Validation: Case Study in a Corba-supported environment", *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'02)*, 2002.
- [7] J. Hörl, B.K. Aichernig, "Validating Voice Communication Requirements Using Lightweight Formal Methods", *IEEE Software*, May/June 2000, pp. 21-27.
- [8] X. Jia, "An Approach to Animating Z Specifications", *Proceedings of 19th Annual International Computer Software and Applications Conference*, Dallas, Texas, USA, 1995.
- [9] E. Kazmierczak, E., P. Dart, L. Stirling, "Verifying Requirements Through Mathematical Modeling and Animation", *International Journal of Software Engineering and Knowledge Engineering* 10(2), 2000, pp. 251-273.
- [10] P. Krutchen, *The Rational Unified Process: An Introduction – 2nd edition*, Reading, Mass, Addison-Wesley, 2000.
- [11] P. G. Neumann, "Only His Only Grammarian Can Only Say What Only He Means", *ACM SIGSOFT Software Engineering Notes* 9(1), 1986, pp. 6.
- [12] OMG, *Object Constraint Language 2 Specification*, October, 2003, available on line: "<http://www.omg.org>".
- [13] OMG, *Unified Modeling Language Specification*, available on line: "<http://www.omg.org>".
- [14] M. Richters, M. Gogolla, "Validating UML Models and OCL Constraints", *Proceedings of UML'2000 - The Unified Modeling Language: Advancing the Standard, Third International Conference*, York, England, 2000.
- [15] A. Sutcliffe, "Scenario-based Requirements Engineering", *Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE'03)*, Los Alamitos CS: IEEE Computer Society Press, 2003.

Specification and Verification of Agent Interaction Protocols

Bo Chen and Samira Sadaoui

Department of Computer Science, University of Regina

Regina, Canada, SK S4S 0A2

{chen112b, sadaouis}@cs.uregina.ca

Abstract

Agent interaction design is one of the principal issues in multi-agent systems. Indeed, the construction of agent interaction protocols (AIP) should integrate theories, methodologies and tools. We propose here a unifying framework that provides a generic agent architecture to be reused as well as a methodology to build and refine AIP specifications in an incremental way. This framework is based on the highly expressive formal language Lotos and its related technologies. It also facilitates validation and verification of AIP specifications using rigorous tools. In addition, we show how to generate an online auction protocol from the framework, and how to verify and simulate this protocol.

1. Introduction

Multi-agent systems (MAS) provide a good means for robust software architectures to develop large-scale commercial and industrial software systems [1]. However, without adequate techniques to support the design process, MAS will not be sufficiently reliable, maintainable, extensible, comprehensible and reusable [16]. Agent interaction protocols (AIP) are used to manage and control agent interaction which is the most important characteristic of MAS. Due to the indeterministic, autonomous and active behaviors of agents, AIP are complex to design and validate. Therefore, formal specification and verification of AIP are necessary to design correct and unambiguous AIP for agent communication in open environments [15].

MAS is essentially concurrent and reactive. It is suitable to apply well-established methods, such as process algebra, temporal logics and finite state machines, to specify, analyze and verify AIP. In this paper, we investigate how to develop correct AIP specifications within a framework. This later is based on the formal specification language Lotos [3] which is an ideal choice for the description, validation and verification of AIP [17, 5, 4]. Lotos combines a process calculus with a data type language. It is executable, modu-

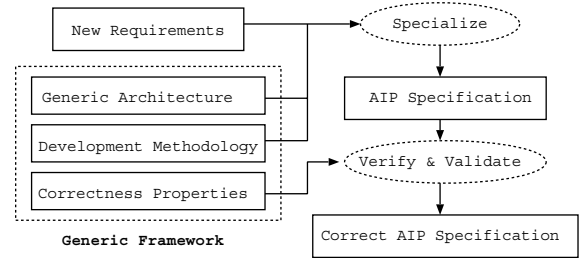


Figure 1. Generic Framework for Building AIP

lar and capable of synchronization between processes. Besides, many tools have been developed for the simulation and verification of Lotos specifications.

As shown in figure 1, the proposed framework not only provides a systematic development methodology for building AIP specifications in an incremental and modular way, but also supports model-checking verification by providing the essential correctness properties (safety, liveness and fairness) that all AIP should satisfy. AIP specifications generated from the framework can also be executed to demonstrate the dynamic behavior of agent communication, and translated into a labeled transition system (LTS) that can be used to manage agent's behaviors in real time.

The paper is organized as follows: Section 2 presents in detail the generic framework for specifying agent protocols. Section 3 introduces a stepwise AIP development methodology. Section 4 explains how to validate and verify an online auction protocol generated from our framework. Section 5 concludes our work with some perspectives.

2. AIP Specification

In table 1, we present the most important AIP entities which can be considered as building blocks for developing any AIP application. These entities are defined through the English auction protocol [4] which describes the interaction

between a seller, an administrator, an auctioneer and several buyers.

Table 1. AIP Entities

Entity	Explanation	Online Auction
Protocol	A set of rules for agent interaction to achieve a goal	English-auction protocol
Purpose	The goal to be achieved through the interaction of agents following the protocol	To sell or buy an item through bidding
Agent	An autonomous and communicative computing entity	An administrator, an auctioneer, a seller and several buyers in a specific auction
Role	A category of agents similar in some aspects	Four roles: Administrator, auctioneer, seller and buyer
Rule	A guideline of behaviors that agents are allowed to act	Online auction rules that each participant has to follow
Communicative Act	To perform actions using speeches (sending messages)	A communicative act is expressed as a message
Message	A structured information that delivers facts and intention of agents	E.g., a seller request-order message contains item description, starting price, bid increment and reserved price
Performative	The intention of sending a message	E.g., call-for-proposal, propose, accept-proposal or refuse-proposal
Message content	A content associated with a performative gives a meaning to a message	E.g., current winner, current price or proposed bid

As shown in figure 2, we model an AIP as a set of communicating Lotos processes that execute concurrently and synchronize on *Send* and *Recv* gates. The constraints on these two gates represent the constraints on sending and receiving messages. They express the protocol mechanism, i.e. the content and order of messages exchanged between agents for negotiation. This model, based on the *social approach* [13, 21], emphasizes the agent collaborative aspects, namely its interactive behaviors. The advantage of this model is that agent interaction can be described even when the internal and mental structure of agents are unclear. Our AIP Lotos specification has around 1000 of lines of code, including 20 data types and 3 processes.

2.1. Message Transportation Service (MTS)

MTS is the core functionality that any protocol needs. It can be reused in all interaction protocol specifications without any modification. It simulates the transferring of a message from a sender to several receivers concurrently. We specify messages as data types which can abstractly represent

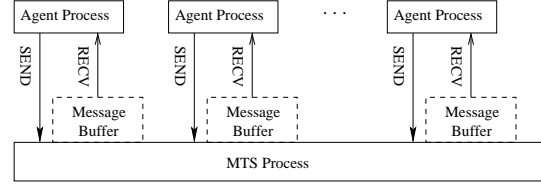


Figure 2. AIP Specification Architecture

message content, being independent of any agent communication language and physical implementation.

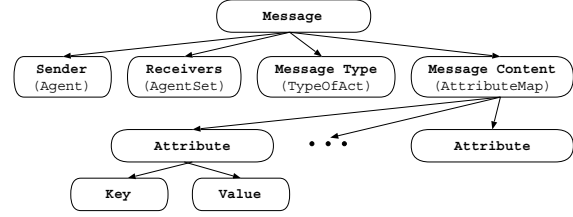


Figure 3. Structure of a Message

As shown in figure 3, we define a message with four data types: a sender (*Agent*), a set of receivers (*AgentSet*), a message type (*TypeOfAct*) and a message content (*AttributeMap*). Some elements are not specified for simplicity such as *conversation ID*, and *protocol* being used. These kinds of information can be set in message content part if necessary. Message content is a map of key and value. This structure is flexible to contain any number and kinds of information.

MTS is modeled by a Lotos process which allows both one-to-one and one-to-many messages transferring. In one-to-many way, a message can be sent to multiple agents at once. Messages can be sent in either synchronous or asynchronous way. When using synchronous communication, the sender waits (blocked) until it makes sure that the message has been taken. When using asynchronous way, the sender does not wait, instead it continues processing immediately after outputting a message.

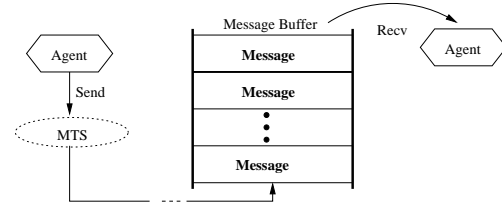


Figure 4. Asyn-Message Transportation

Agents usually use asynchronous point-to-point message communication [8]. We use the two gates *Send* and *Recv* as

well as a set of *message buffers* to describe asynchronous message exchanging, as shown in figure 4. Each message buffer is associated with an agent. All message buffers are combined as a map structure managed by MTS. Asynchronous message exchanging is realized through two synchronization steps given as follows:

- First, complete the synchronization between a sender and MTS. MTS manages all the buffers associated with different agents. A message will be inserted into different buffers of each of its receivers.
- Second, complete the synchronization between the receiver and MTS. In this step, an agent extracts a message and removes it from the buffer.

2.2. Agent Processes

Every agent process has a *SessionData* to control its message outputs. As shown in figure 5, *SessionData* has three items: an identifier (*Agent*), a state (*AgentState*) and attributes (*AttributeMap*). *AgentState* describes the current agent state. A state consists of one or more *tokens*. A message type can be used as a token by applying operations *S* and *R*: *S* means “after sending a message”; *R* means “after receiving a message”. In each state, an agent can only send out some specified types of messages. From the beginning to the ending of an interaction, every agent should at least has an initial and a terminal state. Entering into a new state is triggered by what messages an agent has received and sent, and sometimes the inner events, such as time-out. *AttributeMap* here is the same as message content. This is a natural representation considering that a message just delivers the knowledge of an agent. *AttributeMap* can represent any number of attributes used by an agent to manage its conversation. For example, *AttributeMap* of an auctioneer agent has the following attributes: administrator, a set of buyers, auction item, seller, starting price, bid increment, reserved price, current bid and current winner.

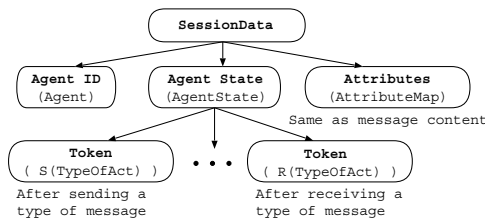


Figure 5. Structure of SessionData

3. Incremental Development of AIP

The design of concurrent systems is a complex task. Hence it is better to construct AIP in an incremental approach also

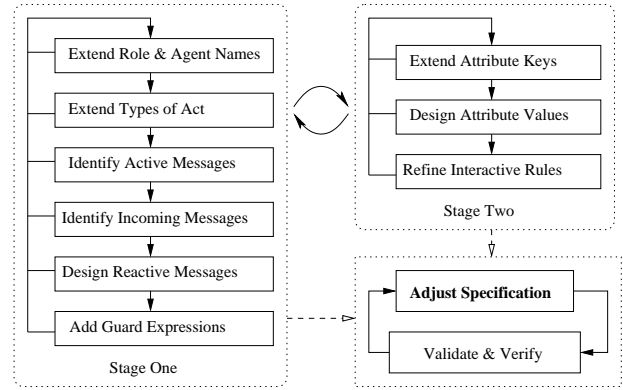


Figure 6. AIP Development Methodology

called step-wise refinement. We divide AIP development into two major stages, each one consists of several steps. All the steps in the first and second stages are illustrated in figure 6 and discussed below.

The first stage decides only the types of communicative acts and transitions among them without message content. One incoming message will correspond to one outgoing message. Message content is not meaningful at this stage which only defines the basic mechanisms of AIP.

1. **Extend Role and Agent Names.** New role and agent names are added. This extension is easy because these data types are defined with constructor operations.
2. **Extend Types of Act (Performative).** New message types are added by extending the data type *TypeOfAct*.
3. **Identify Active Messages.** Identify what types of messages a role can send out actively based on its internal states and events. These actions are not triggered immediately by the received messages.
4. **Identify Incoming Messages.** Identify what types of messages a role can receive.
5. **Design Reactive Messages.** Identify what types of messages a role has to reply when receiving a message.
6. **Add Guard Expressions.** In order to remove some indeterministic choices, Lotos guard expressions are defined using the attributes of *SessionData*.

In the second stage, more controls will be considered when message content is added into the specification. This stage will produce the complete specification of a protocol.

1. **Extend Attribute Keys.** The keys of attributes are extended to add new attributes into agent *SessionData* and *Message*. For instance, an auction message may have the keys: *start_price*, *bid_amount*, *bid_increment*, etc.

2. **Design Attribute Values.** A value can be of any sort: *Agent*, *AgentSet*, *Money*, etc. These keys and values together define meaningful *SessionData* and *Message*.
3. **Refine Interactive Rules.** Given the message content, Lotos guard expressions are refined to represent more subtle conditions and constraints for sending messages.

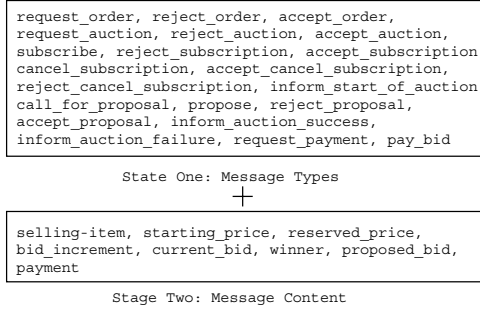


Figure 7. English Auction Message

Our development approach builds a protocol step by step. In each step, more details are added into the specification. Refinement will continue until all the mechanisms of a protocol are completely specified. Thus, a complex protocol can be generated using our framework. Our framework has been successfully experimented through the English auction protocol. The message types and message content in stage one and two are illustrated in figure 7. The resulting Lotos specification has around 1600 of lines of code, including 23 data types and 5 processes. We also note that any auction type (Dutch, Vickrey, Yankee, First-Price ...) can also be generated from the framework.

4. Validation and Verification

Many tools have been developed for the validation and verification of Lotos specifications. One of the powerful tools is CADP [10] illustrated in figure 8. CADP is an engineering tool that assists the user through the design process: compilation, interactive and goal-oriented simulation, test generation for protocol implementation, rapid prototyping by generating the C code which can be embedded in real applications, and most important, CADP can efficiently perform verification by equivalence and temporal logic model-checking.

With the interactive simulation, we can trace and monitor all the possible execution sequences. CADP can also generate all the scenarios that satisfy a user-defined goal. For instance, the scenarios of an auction-failure goal are: the auction ends and best bid is lower than the reserved price; the order request is refused by the administrator agent; the

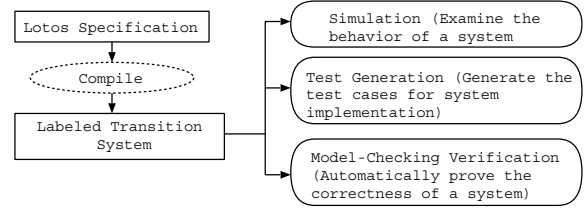


Figure 8. CADP ToolBox

auction ends and no buyer has subscribed. These scenarios are illustrated below:

- SEND !MSG (SELLER, @(ADMINISTRATOR, <>), REQUEST_ORDER, @(&(K.BIDINCREMENT, 1), @(&(K.RESERVEPRICE, 2), @(&(K.STARTPRICE, 1), <>))))
(* Seller requests administrator to sell an item and starting price is 1, bid increment is 1, and reserved price is 2. *)
- RECV !ADMINISTRATOR !MSG (SELLER, @(ADMINISTRATOR, <>), REQUEST_ORDER, @(&(K.BIDINCREMENT, 1), @(&(K.RESERVEPRICE, 2), @(&(K.STARTPRICE, 1), <>))))
(* Administrator receives the order request. *)
- SEND !MSG (ADMINISTRATOR, @(SELLER, <>), ACCEPT_ORDER, <>) (*Administrator sends to seller an acknowledge to accept the order*)
- SEND !MSG (ADMINISTRATOR, @(AUCTIONEER, <>), REQUEST_AUCTION, @(&(K.BIDINCREMENT, 1), @(&(K.RESERVEPRICE, 2), @(&(K.STARTPRICE, 1), <>))))
(* Administrator submits the order to auctioneer for processing *)
- RECV !AUCTIONEER !ADMINISTRATOR !MSG (ADMINISTRATOR, @(AUCTIONEER, <>), REQUEST_AUCTION, @(&(K.BIDINCREMENT, 1), @(&(K.RESERVEPRICE, 2), @(&(K.STARTPRICE, 1), <>))))
(* Auctioneer receives the auction request *)
- SEND !MSG (AUCTIONEER, @(ADMINISTRATOR, <>), ACCEPT_AUCTION, <>)
(* Auctioneer sends an acknowledge to administrator *)
- i (* Auction starts, internal action *)
- i (* Auction ends and no buyer has joined this auction, internal action *)
- SEND !MSG (AUCTIONEER, @(ADMINISTRATOR, @(SELLER, <>)), INFORM_AUCTION_FAILURE, <>)
(* Auctioneer informs administrator and seller about the auction failure *)

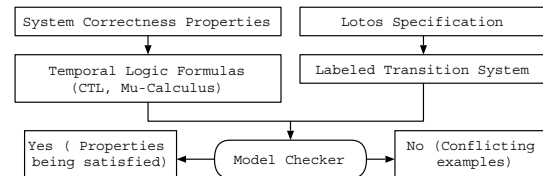


Figure 9. Model-Checking Verification

Model-checking is more powerful than simulation. It can be used to prove the correctness of a protocol. To do the model-checking, we first have to decide all the required correctness properties, then we encode them as temporal logic formulas using *CTL* and *Mu-calculus* [19], as illustrated in

figure 9. Usually there are three kinds of correctness properties: safety, liveness and fairness, described below.

- Safety properties express that something bad never happens in a system. For example, there is no deadlock (progress is no more possible) or livelock (the system enters a process and cannot leave) in the specification, or some actions should always occur before other actions. To facilitate the verification, we can define some temporal logic macro expressions. For example, the order of two actions A and B can be expressed using a user-defined macro: $before(A, B) = not([(not (B))^* . (A)] false)$. In AIP, an agent usually can not perform some actions or enter into some states without being triggered by incoming messages. These important properties of AIP can be verified using the *before* macro.
- Liveness properties express that something good eventually happens. A liveness property requires that at least one sequence of messages in the protocol satisfies the temporal formula. An example is the absence of starvation (eventually each process is granted a resource). Another one is accessibility requiring that a conversation should lead to a desired state from the initial state [15].
- Fairness properties are needed when several processes compete for a resource. They require that each action has the infinite opportunity to be performed when it is enabled infinitely. For instance, the property that an action A will be fairly reached initially can be expressed as: $[(not "A")^*] < true * . "A" > true$.

AIP properties can also be directly modelled in Lotos. The observational equivalence between property specification and protocol specification is verified automatically. The observational equivalence is also useful to check the compliance between refined AIP specifications and generic ones.

We have proved that the generated English auction specification is deadlock and livelock free. In addition, we have proved some correctness properties specific to the auction protocol, including:

- A proposal cannot be accepted by auctioneer unless the buyer's subscription has been accepted (safety property).
- An unsubscribed buyer cannot receive a call-for-proposal (safety property).
- There exists an execution sequence that leads to an auction-success (liveness property).
- After a subscription, a buyer can receive a call-for-proposal (liveness property).
- A buyer has the chance to win if no other buyers want to pay more than (maybe as much as) its bid (fairness property).

We also note that when the auction specification in-

cludes for instance two buyers, the model checker generates 3228732 states and 10733000 transitions using a Sun Ultra Sparc Station (1Ghz of CPU and 8GB of RAM memory).

5. Conclusion and Perspectives

As discussed in this paper, our framework provides a flexible generic architecture to correctly and completely specify AIP in appropriate abstraction levels. This architecture expresses almost all aspects of agent interaction, and supports cases that involve groups of agents such as protocols of bidding, election and voting. The message structure is flexible, meaningful and domain-independent. Synchronous and asynchronous communication can be naturally expressed. AIP Lotos specifications can also be simulated to observe the dynamic behavior allowed by AIP, thus improving people's understanding of complex AIP. Most important, we can use model-checking tools to automatically verify the properties of AIP specifications and prove their correctness.

Our future work is to build a tool that assists the construction of AIP applications based on the generic framework. Also, the implementation of the specification protocols are necessary to complete the whole lifecycle software development. Since protocols are considered as reusable components [8], we also want to investigate how to design advanced AIP applications by reusing and composing existing ones. Lotos is a highly expressive language. However, it has some limitations, such as that it cannot specify quantitative time and exception handling. We want to improve this generic framework by using ELotos [14]. ELotos removes some limitations and provides better structuring mechanisms, such as modularity, interface and other user-friendly features. We would like to apply ELotos in such area but there is no supporting tools. This is why we are currently developing a Java simulator for ELotos.

References

- [1] T. Arai and F. Stolzenburg. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In *Proceedings of The First International Joint Conference On Autonomous Agents And Multi-agent Systems*, pages 11-18. ACM Press, 2000.
- [2] F. Bergenti and A. Ricci. Three Approaches to the Coordination of Multiagent Systems. In Y. Demazeau, editor, *2002 ACM Symposium on Applied Computing (SAC'02)*, pages 367-372, Madrid, Spain, 2002.
- [3] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14(1):25-59, January 1988.

- [4] B. Chen and S. Sadaoui. Simulation and validation of a dynamic online auction. In *7th IASTED International Conference on Software Engineering and Applications*, 2003.
- [5] M. A. Cornejo, H. Garavel, R. Mateescu, and N. D. Palma. Specification and verification of a dynamic re-configuration protocol for agent-based applications. In *DAIS*, pages 229–244, 2001.
- [6] R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversations with colored petri nets. In *Working Notes of the Workshop on Specifying and Implementing Conversation Policies*, pages 59–66, Seattle, Washington, May 1999.
- [7] R. Maria d. C. Andrade. *Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems*. PhD thesis, University of Ottawa, 2001.
- [8] M. d’Inverno, D. Kinny, and M. Luck. Interaction protocols in agentis. In *Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 261–268, 1998.
- [9] M. Wooldridge, N. R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents’99)*, pages 69–76, Seattle, WA, USA, 1999. ACM Press.
- [10] J.-C. Fernandez, H. Garavel, A. Kerbrat, and L. Mounier. CADP: a protocol validation and verification toolbox. 1996.
- [11] L. F. Pires, and W. L. de Souza. Stepwise refinement design example using LOTOS. In Juan Quemada, Jose A. Mañas, and Enrique Vázquez, editors, *Proc. Formal Description Techniques III*. North-Holland, Amsterdam, Netherlands, November 1990.
- [12] FIPA ACL Message, <http://www.fipa.org/specs/fipa00061>, 2001.
- [13] N. Fornara and M. Colombetti. Defining interaction protocols using a commitment-based agent communication language. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 520–527. ACM Press, 2003.
- [14] H. Garavel and M. Sighireanu. Towards a second generation of Formal Description Techniques – Rationale for the design of E-LOTOS. In Jan-Friso Groote, Bas Luttik, and Jos van Wamel, editors, *Proc. 3rd. International Workshop on Formal Methods for Industrial Critical Systems*, pages 187–230, Amsterdam, Netherlands, May 1998. University of Nantes.
- [15] H. Mazouzi, A. E. F. Seghrouchni, and S. Haddad. Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 517–526. ACM Press, 2002.
- [16] D. Kinny and Mi. Georgeff. Modelling and Design of Multi-Agent Systems. In *Intelligent Agents III: Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Budapest, Hungary, 1996.
- [17] J.-L. Koning. Algorithms for translating interaction protocols into a formal description. In *IEEE International Conference on Systems, Man and Cybernetics*, Tokyo, Japan, October 1999.
- [18] J.-L. Koning. Designing and testing negotiation protocols for electronic commerce applications. In Frank Dignum and Carles Sierra, editors, *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Computer Science*, pages 34–60. Springer, 2001.
- [19] R. Mateescu and M. Sighireanu. Efficient on-the-fly model-checking for regular alternation-free mu-calculus. In *Proceedings of the 5th International Workshop on Formal Methods for Industrial Critical Systems FMICS’2000*, Berlin, Germany, April 2000.
- [20] J. Odell, H. Parunak, and B. Bauer. Extending UM for agents. In *Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, 2000.
- [21] M. Viroli and A. Omicini. Specifying agent observable behaviour. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 712–720. ACM Press, 2002.
- [22] C. A. Vissers, G. Scollo, M. v. Sinderen, and Ed Brinksma. Specification styles in distributed systems design and verification. In *Selected papers of the 6th International conference on Logic programming*, pages 179–206. Elsevier Science Publishers B. V., 1991.
- [23] M. Wooldridge. Verifiable Semantics for Agent Communication Languages. In Y. Demazeau, editor, *Proceedings of the Third International Conference on Multi-Agent Systems*, pages 349–356, Paris, France, 1998. IEEE Press.

Supporting the Requirements Prioritization Process. A Machine Learning approach.

Paolo Avesani¹, Cinzia Bazzanella^{1,2}, Anna Perini¹, Angelo Susi¹

¹ITC-IRST, Via Sommarive 18, I-38050, Povo-Trento, Italy

{avesani,perini,susi}@irst.itc.it

²University of Trento, Via Sommarive 14, I-38050, Povo-Trento, Italy

{bazzanella}@irst.itc.it

Abstract

Requirements prioritization plays a key role in the requirements engineering process, in particular with respect to critical tasks such as requirements negotiation and software release planning.

This paper presents a novel framework which is based on a requirements prioritization process that interleaves human and machine activities, enabling for an accurate prioritization of requirements. Similarly to the Analytic Hierarchy Process (AHP) method, our framework adopts an elicitation process based on the acquisition of pairwise preferences. Differently from AHP, where scalability is a big issue, the framework enables a prioritization process even over a large set of requirements, thanks to the exploitation of machine learning techniques that induce requirements ranking approximations at run time, and to the use of a boolean metrics. Moreover the new approach allows to reduce the bias of a dominance hierarchy, a strategy introduced by AHP to deal with the scalability issue.

The paper describes also a methodology for the experimental evaluation of the framework and discusses the results of a first set of experiments designed on a real case-study which shows that an high accuracy in the final ranking can be obtained within a limited elicitation effort.

1. Introduction

Requirements prioritization has been pointed out as a relevant research area in requirements engineering, calling for the definition of effective methods and techniques that enable to rank a whole set of requirements, according to relevant criteria, such as business goals (e.g. customer value) or technical features (e.g. development cost) [15].

Prioritizing requirements can be seen as the process of deriving an order relation on a given set of requirements, with the ultimate goal of obtaining a shared rationale for partitioning them into subsequent product releases.

Several approaches have been recently proposed [8, 7, 9, 10, 14], which adopts a common model for the requirements prioritization process, based on the following three

steps: (i) selection of one or more prioritization criteria (or prioritization features) among business goals and technical features; (ii) acquisition of a requirements ordering according to a specific criterion from one or more stakeholders (e.g. customers, users, project manager); (iii) composition of the acquired orderings into a final one based upon an appropriate composition schema. These approaches tend to focus on how to choose the most relevant criteria and on how to combine them, while giving minor emphasis to the acquisition of the ranks according to a given criterion.

In our approach, we address the problem of supporting the acquisition of requirements ordering according to prioritization features, (i.e. step (ii)), taking into account critical issues related to this step, such as how to manage stakeholder bias and how to maintain the costs of the prioritization process lower than the resulting benefits.

In order to reduce the risk of ambiguity of the stakeholder judgment, we devise the preference acquisition task as a pairwise comparison on the set of requirements, adopting an approach similar to the one proposed by Saaty [13] within the Analytic Hierarchy Process (AHP) method. Although a pairwise prioritization approach is successful in reducing the acquisition error, the total amount of information that has to be acquired from the stakeholder increases quadratically with the number of requirements, making scalability a critical issue. This problem has been faced by introducing a dominance hierarchy [13], but this may introduce a bias on the resulting ranking.

In our approach we exploit machine learning techniques to reduce the elicitation effort by approximating part of the pairwise preferences. The approximation step computes an estimate of unknown preferences looking at the other ranks acquired according to predefined prioritization criteria.

Moreover, we adopt a boolean metrics to lower the human effort associated to the requirements evaluation and we prove that it can be effective as much as multi values metrics, as far as a large set of requirements has to be prioritized [4].

The resulting framework supports the ranking process by providing a mixed-initiative strategy that combines human

effort and computer support to accomplish the task. In this paper we describe our framework and propose a methodology for the experimental evaluation of its effectiveness. This methodology exploits a web application to elicit requirements prioritization. We are conducting a set of experiments with a group of students of the computer science faculty, using a case study extracted from a real application. The experimental results of a first set of experiments are really promising because they show that we can obtain an accurate requirements ranking with a limited elicitation effort.

The paper is structured as follows. In Section 2 we present the framework and the machine learning techniques that it is based on. In Section 3 we describe the evaluation methodology and discuss experimental results. Related work are briefly discussed in Section 4. Finally, conclusions and future work are presented in Section 5.

2. Our approach

We propose a framework that adopts the AHP technique of pairwise prioritization and exploits machine learning techniques to overcome AHP limits.

AHP [13] is a multiple criteria decision making technique that allows decision makers to take into account trade-offs between attributes.

In applying AHP to requirements prioritization, the first step is the selection of a given set of requirements, called also *alternatives*. The second step is the definition of a dominance hierarchy for the criteria that have to be taken into account in the evaluation process: at the top of the dominance hierarchy are the objectives from a managerial point of view, at the lowest level, a list of alternatives. In the third step, given a criterion, a pairwise comparison matrix is built. The rows and columns of this matrix represent the set of requirements under investigation. Half of the matrix elements are assigned with an integer belonging to the interval $[0 \dots 9]$ which represents a qualitative measure of the preference relation (e.g. the requirement *A* is “equally important” than requirement *B* respect to the given criteria, corresponds to the value 1, the requirement *A* is “essentially more important” than requirement *B* corresponds to the value 5). At the end of the voting process a total order is synthesized through the computation of a vector of weights that specifies the rank of each requirement. This step is repeated for all the criteria. The vectors of weights and the related ranks represent the different points of view according to the predefined criteria. The last step is the synthesis of a global rank that composes the different orderings. An analogous preference elicitation process is performed by filling a matrix where rows and columns represent the different criteria taken into account. In such a way a rank on the criteria is obtained and is used to compute a weighted composition of the different order relations defined over the set of alternatives.

Among the main limits of AHP, the dramatic growth of

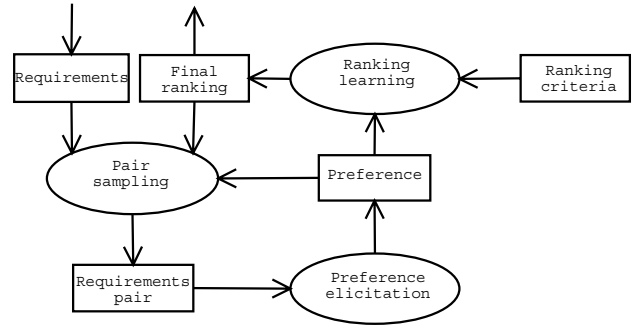


Figure 1. The basic iteration of the requirements prioritization process.

the number of comparisons needed as long as the number of candidate requirements increases. While the adoption of a dominance hierarchy has been proposed to handle this phenomenon, this can introduce a bias in the priority elicitation.

Another critical aspect of the AHP technique concerns the ten value rating scale, which may cause problems of cognitive overload (in specifying one value preference out of ten rates), as well as of semantic ambiguity (e.g. different stakeholders perceive in a different way the difference between “more important” and “essentially more important”).

In our approach we adopt the AHP pairwise comparison technique, using a binary rating, and we exploit machine learning techniques to approximate part of the pairwise preferences in order to reduce the elicitation effort avoiding the use of a dominance hierarchy.

Our framework supports an iterative process for requirements priority elicitation that can handle single and multiple evaluators (stakeholders) and different criteria (both business goals and technical parameters). In the following, we illustrate it considering the case of a stakeholder who collaborates to the prioritization of a set (of cardinality n) of system requirements, given a ranking criteria. We suppose also that the requirements have been already ranked respect to technical characteristics (e.g. the cost or the effort to realize a requirement, or the penalty cost in case the requirement will be not included in the system release).

Figure 1, depicts the basic process that the evaluator undertakes. The types of data involved in the process are depicted as rectangles, namely: **Requirements** represent data in input to the process, that is the finite collection of requirements that have to be ranked; **Requirements pair** is a pair of candidate requirements whose relative preference is to be specified; **Preference** is the order relation between two alternative requirements elicited from the stakeholder. The preference is formulated as a boolean choice on a pair; **Ranking criteria** are a collection of order relations that represents ordering induced by other criteria (e.g. the cost for the realization of the requirements, the estimated utility) defined on the initial set of requirements; **Final ranking** represents the resulting preference structure on the set of requirements. This final ranking which results from the

output of the process represents an approximation of the exact ranking. Notice that this ranking may become the input to a further iteration of the process.

The steps of the basic process iteration are depicted as ovals in Figure 1, they are:

1. Pair sampling

An automated procedure selects from the requirements repository a pair of requirements and submits it to the stakeholder who can judge their relative priority. Notice that in this step, the selection of a pair takes into account information on the current available rankings (this information is stored in the data **Preference**, see the arrows between **Preference** and **Pair sampling** in Figure 1);

2. Preference elicitation

This represents a mixed initiative step in the process: given a pair of requirements the stakeholder chooses which one is to be preferred with respect to the current criterion;

3. Ranking learning

Given a partial elicitation of the user preferences, a learning algorithm produces an approximation of the unknown preferences and a ranking of the whole set of requirements is derived.

If the result of the learning step is considered accurate enough or the end user has been overloaded, the iteration halts and the latest approximated rank is given as output; otherwise another cycle of the loop is carried on. It is to be noticed that the first and the third steps are automated while the second step is in charge to the stakeholder. The model is characterized by the fact that the preference elicitation is monotonic (i.e. the user does not see the same pair twice). It could be helpful to remind that such a method aims at obtaining a lower human effort/elicitation, while increasing accuracy of the approximation.

2.1. The learning algorithm

The **Ranking learning** step produces an approximation of a preference structure, exploiting machine learning techniques and in particular the boosting approach described in [6]. In the following we give a brief description of the problem that we handle with the boosting approach and of the algorithm.

We have a finite set of requirements $Req = \{req_0, \dots, req_n\}$. The ranking criteria $F = (f_1, \dots, f_m)$ defined as a finite set of m functions that describe the single requirement inducing an ordering on the set Req , where $f_j : Req \rightarrow \mathbb{R}$ ($\mathbb{R} = \mathbb{R} \cup \{\perp\}$) and the interpretation of the inequality $f_j(req_0) > f_j(req_1)$ means that req_0 is ranked above req_1 by f_j and $f_j(req) = \perp$ if req is unranked by the functions in F . As already mentioned, they can represent business goals, such as user satisfaction or more technical aspects, such a measure of the estimated cost for the implementation of the requirements or the penalty cost for a not complete implementation of a given requirement.

The target ranking represents the ideal requirements ordering expressed by a stakeholder and is defined as the function $\Phi : Req \times Req \rightarrow \{-1, 0, 1\}$ where $\Phi(req_0, req_1) = 1$ means that req_1 be ranked above req_0 , $\Phi(req_0, req_1) = -1$ means that req_0 be ranked above req_1 , and $\Phi(req_0, req_1) = 0$ indicates that there is no preference between req_0 and req_1 (we assume $\Phi(req, req) = 0$ and $\Phi(req_0, req_1) = -\Phi(req_1, req_0)$ for all $req, req_0, req_1 \in Req$). Related to the Φ we also define a density function $D : Req \times Req \rightarrow \mathbb{R}$ such that $D(req_0, req_1) = \gamma \cdot \max(\{0, \Phi(req_0, req_1)\})$ setting to 0 all negative entries of Φ ; γ is a positive constant chosen such that D is a distribution, satisfying the normalization property¹ $\sum_{req_0, req_1} D(req_0, req_1) = 1$.

The goal of the learning step is to produce a ranking of all requirements in Req . This ranking is represented in the form of a function $H : Req \rightarrow \mathbb{R}$ where req_1 is ranked higher than req_0 by H if $H(req_1) > H(req_0)$. The function H represents the approximate ordering of Req induced by the feedback function Φ using the information from the set of features F .

In our framework, the function H is computed by a learning procedure based on boosting method. Boosting iteratively combines many learners, usually weak learners, into a final learner. For example a ranking prediction function H can be obtained as a linear combination of many simple functions that define a simple ranking rule as a partition over the given set of requirements.

That is we compute the function H in the form of a linear combination of partial order functions $h_t : Req \rightarrow \mathbb{R}$ (weak rules) with a set of coefficients $\alpha = \{\alpha_1, \dots, \alpha_t, \dots\}$. The algorithm that computes H performs T iterations; it takes as input the initial distribution D and the set of functions F .

The basic iteration performs the three steps described below.

- Compute a partial order h_t of the elements in Req taking into account both the user feedback function Φ and the orderings induced by the functions in F .
- Compute a value for the parameter α_t . This value is a measure of the accuracy of the partial order h_t respect to the final order H .
- Compute a new distribution D over the set of pairs already evaluated by the evaluator, which is passed, on the next iteration, to the procedure that computes the partial order h . The basic intuition is that the distribution D represents the portion of relations where the algorithm fails to produce an accurate prediction. Therefore the information provided by the distribution D is given in input even to the pair sampling policy. Pairs where the priority is supposed to be less accurate will be presented to the users for the next step of preference elicitation.

The number of iterations can be fixed a-priori or the algorithm stops when a stable ordering configuration has been found. More details on the algorithm can be found in [3].

¹Notice that $\Phi(req_0, req_1) = 0$ means that the pair hasn't been proposed to stakeholder, so this three valued function allows to represent the boolean choice of the stakeholder.

3. Empirical Evaluation

The development of a methodology for evaluating requirement prioritization techniques poses interesting problems, such as how to define measurable factors against which to conduct the evaluation, how to organize the experiments; how to choose, train and monitor the evaluators.

We considered two evaluation approaches, that we refer as off-line and on-line evaluations. The former doesn't involve the real users but relies on simulations of the requirement prioritization process. The latter is based on a test on the field taking into account a real setting of requirement engineering process and involving the stakeholders in the assessment of the methodology. The off-line approach is usually easier because the simulation of the prioritization process allows to define in advance what is the correct solution, i.e. the target priority index. Given this premise, it is possible to introduce a measure of *disagreement* between the target priority relation and the ranking relation obtained by the proposed methodology. Disagreement is defined as the ratio between the disalignment between the two relations with respect to the total number of pairwise precedence relationships. In [4] we discussed the results of off-line evaluations, showing that, even though it may appear counterintuitive, a boolean prioritization metrics can be more effective than a multi-value metrics, as far as the number of requirements become larger and larger, and that our framework allows to maintain the costs of the prioritization process lower than its benefits.

The on-line approach is usually harder because it is not known at all the value of *correct* priority index for a specific requirement prioritization problem. What happens for real is the following: once a prioritization methodology is applied we obtain as solution a priority index but we lack the opportunity to compare it with the target priority index. There is a paradox: if we were able to acquire the *correct* solution we would already have the method to compute the priority index. Below we propose an approach to empirical evaluation suitable to overcome these drawbacks.

We set up an on-line evaluation methodology extracting a case study from the development of a real application called CoCoA, Compilation Compiler Advisor [1]. CoCoA² is a web application to deliver a personalization service for audio compilation. Up to now more than 50.000 users has edited and downloaded more than 50.000 compilations with a repository of more than 11.000 mp3 tracks.

From the documentation of the project we extracted a subset of 40 requirements defined at the early stage to describe the CoCoA system. Figure 2 gives an excerpt of requirement definition which consists basically of a unique identifier that allows to manage a non ambiguous reference; the informal and verbose description; a set of ranking features, for example the requirement effort, which can be used to induce an order relation with respect to this criterion.

²You may have a trial at the following address: <http://cocoa.itc.it>

```
<requirement>
<idR-#33</id>
<title>Audio Track Download</title>
<type>functional</type>
<effort metric="mm">0.25</effort>
<relevance metric=[0,9]>??</relevance>
<description>
    Once the compilation is completed it has to be supported the operation
    of download for each single track. No constraints should be applied
    to the track download order.
</description>
</requirement>
...
<requirement>
<idR-#41</id>
<title>Track Recommendation</title>
<type>functional</type>
<effort metric="mm">0.50</effort>
<relevance metric=[0,9]>??</relevance>
<description>
    Anytime, the user should have the opportunity of receiving a recom-
    mendation on suitable tracks to complete her/his compilation under
    development. No additional personal information has to be provided to
    the system to take advantage of recommendation service.
</description>
</requirement>
```

Figure 2. Requirement Excerpt. A couple of examples of requirements that have been used in the experimentation.

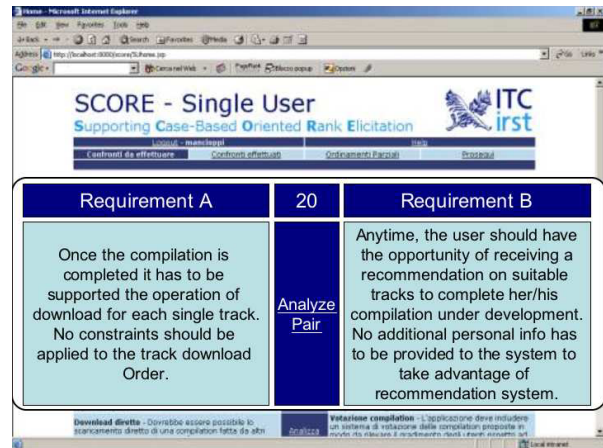


Figure 3. Snapshot GUI. A snapshot of the graphical user interface related to the agenda of pairwise comparison.

In order to apply the methodology we developed a web application which support a distributed use of the framework. The goal was to support the pairwise priority elicitation by distributed stakeholders (the users of the system). The Figure 3 shows a snapshot of the graphical user interface.

The empirical evaluation was restricted to the elicitation of a specific ranking criteria concerned with user relevance. Five persons has been selected as representative of end users. The goal of our experiment was to assess whether our methodology is effective in supporting the acquisition of one of the ranking criterias that take part in the definition of a general priority index.

The typical session has been designed as follows. A collection of pairwise alternatives was presented to the end

uses. They were invited to select a pair, to assess the relative priority between the two respective requirements, to elicit the priority relation among them. No additional effort was required, just a boolean preference. The interactive session was very simple indeed, no specific training or tutorial was required. Anyway, it is important to remark that no free riding over the requirements or the related priority index was allowed.

As mentioned before our stakeholders, in the specific context, were the end users representative. Each of the five persons involved in the experimentation, autonomously attended the simple process of requirement prioritization. The distributed design of the experiment allows us to avoid any supervision activity during the process, so no additional bias has been introduced.

The requirement prioritization process has been configured with 40 requirements. Each user has been invited to analyze 60 comparisons and to elicit pairwise priority relations accordingly. A first agenda with 20 bipartite pairs has been proposed to the user that since the beginning had an overview of the requirement set. The total amount of elicitation was only the 8% of the all pairwise relationships that should have to be elicited.

After the elicitation process supported by our method we obtained five priority indexes, one for each user. Each priority index was partially based on elicited preferences and partially (mostly) on approximated values.

At this stage we meet the problem of quality assessment due to the lack of knowledge of the target priority index. We proceeded as follows. The intuition is to refer to the variance among the different users to select a subsample of pairwise priority relations to be tested with real users.

Let us remember that all the potential pairwise priority relations are $O(n^2/2)$ and the known relations are only $3/2n$, where $n = 40$ is the cardinality of requirement set. Therefore the unknown pairwise priority relations to be tested are too many for an exhaustive assessment. For this reason we need an heuristic to select a subset of them.

We first computed the cumulative agreement among the different users. For each rank position, from 1 to 40, referred as the k -th position, we computed the relative percentage of requirements that the users assigned the same order. We obtained three partitions of maximal agreement among the evaluators. Within each partition we computed the variance of each user with respect to such category. Given, for example, the first category, that was identified in the range $[1 \dots 12]$ (i.e. $k = 12$), we know that only part of the requirements ordered in the first twelve positions by user 1, have been placed in the same category by all other users. We focussed our attention on the subset of requirements that a given users placed in the first 12 position differently from other users. Therefore for each of such requirements we synthesized a pair with the requirement at 12-th position. We did the same for the subsequent categories obtaining two subsets of pairs: the former to test a forward prioritization error, the latter to test the backward prioritization error.

	Prioritization Error		
	FW (%)	BW (%)	tot (%)
user 1	5	0	5
user 2	0	0	0
user 3	2	0	2
user 4	5	5	10
user 5	5	2	7
average	3	1	4

Table 1. Experimental Results.

Forward prioritization error refers to requirements that are erroneously ranked with higher priority. Backward prioritization error, on the contrary, refers to underestimate priority. The rationality of such a heuristic is to focus the testing over the variance with the goal of assessing whether it was an approximation error or only a specificity of the user.

The ultimate stage was designed for testing purpose. It consisted in a supplementary session of pairwise priority elicitation performed by the end users and it has been conducted as a completely blind process.

Finally we were able to directly compare on a subset of the priority relations the real priority values provided by the users and the priority values approximated by the machine. Table 1 summarizes the results of the on-line experimental evaluation.

On average we obtained that with only a 8% of elicitation effort we can achieve a prioritization results approximately 96% accurate. A threshold of 4% of error can be considered quite reasonable and not meaningful compared with the noise related to such a kind of process.

Less positive is the variance of the error among different users. For user 2 we succeeded in reducing to 0% the error while for the user 4 we missed 10%. This drawback can be reduced promoting a collaborative setup of the methodology that aims to share on the fly the elicitation effort of different users.

The difference between the forward and backward errors is a side effect of the evaluation approach since the learning techniques don't introduce any bias to favour upper position rather than lower ones. The amount of pairs selected to assess forward and backward errors differs, while the relative error over these amounts is the same.

4. Related work

Recent approaches to requirements prioritization exploits different multi-criteria decision making techniques, such as Analytic Hierarchy Process (AHP) [13], Multi Criteria Decision Aid (MCDA) [11], SMART [5] and Quality Function Deployment (QFD) [2]. Among them the cost-value approach [8], the Multi-criteria Preference Analysis Requirements Negotiation (MPARN) [7], and Quantitative WinWin [12].

Main limits of these techniques are attributed to the strong assumptions they adopt, such as, the completeness

and certainty of the set of requirements to be evaluated and the plausibility of a rating scale based on discrete categories. Moreover, they seem to be inadequate in handling the following relevant issues: cost of the elicitation process; subjectivity of the stakeholders opinions; dependencies among requirements; requirements volatility.

In this paper, we addressed in particular the first two problems. Other frameworks attempt to manage them integrating different decision making techniques as well as methods for the identification of relevant criteria for requirements prioritization derived from other disciplines, (e.g. portfolio-based reasoning) [9, 8, 7, 10, 14].

Relevant to the work described in this paper are AHP-based methods, such as the Soft Requirements Negotiator (SRN) [10]. The SRN method aims at addressing the incompleteness and uncertainty of the initial set of requirements to be prioritized and for this reason integrates AHP, MCDA and simulation techniques for the estimation of quantitative ranking features. MCDA techniques are exploited with the attempt to deal with incomplete information, and in particular to support the selection of balancing “for” and “against” arguments for a given requirement.

The method rests on a two phase process: (i) the qualitative analysis phase consisting in the acquisition of the stakeholder preferences, that aims at partitioning the requirements into three categories; (ii) the quantitative analysis phase where quantitative data referred to cost and value are used to compute the set of most promising requirement rankings. In particular, during phase (i), the evaluator (a stakeholder) is asked to give a rank to the requirement on the basis of a three values scale ($\{-, 0, +\}$), with the following meaning: “+” if the stakeholder considers a requirement important with respect to the selected criterion; “-” if it is not important; “0” in case of a neutral judgment. This approach gives as a result a partial ordering of the elements. We think that our techniques could be fruitfully applied in phase (i) of the Ruhe framework.

5. Conclusion and future work

In this paper we presented a novel framework for requirements prioritization, which adopts an elicitation process based on the acquisition of pairwise preferences. Differently from AHP, where scalability is a big issue, our approach enables a prioritization process even over a large set of requirements, thanks to the exploitation of machine learning techniques that induce requirements ranking approximations from the acquired data, and to the use of a boolean metrics. Moreover the new approach allows to reduce the bias of a dominance hierarchy, a strategy introduced by AHP to deal with the scalability issue.

A methodology for the experimental evaluation of the framework has been described. This methodology is currently used with a group of students to perform a set of tests designed on a real case-study. The results of a first set of experiments showed that a high accuracy of the fi-

nal requirements ranking (96%) can be obtained with a low elicitation effort (less than 8% of the possible requirements pairs). On-going experiments are aimed at comparing different requirements prioritization methods.

We believe that the current results are promising and we are going to further investigate the framework addressing other critical issue of requirements prioritization such as the negotiation among many viewpoints of different stakeholders; handling of requirements dependencies; “anytime” prioritization when new unexpected requirements are added.

References

- [1] S. Aguzzoli, P. Avesani, and P. Massa. Collaborative Case-Based Recommender Systems. In *Proceedings of European Conference on Case-Based Reasoning (ECCBR 2002)*, LNAI, Aberdeen, Scotland, 2002. Springer-Verlag.
- [2] Y. Akao. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Productivity Press, 1988.
- [3] P. Avesani, S. Ferrari, and A. Susi. Case-Based Ranking for Decision Support Systems. In *Proceedings of IC-CBR 2003*, number 2689 in LNAI, pages 35 – 49. Springer-Verlag, 2003.
- [4] P. Avesani, A. Perini, and A. Susi. Prioritizing requirements via case-based ranking, 2004. IRST Technical Report 1301-04.
- [5] W. Edwards and F. Baron. *Smarts and smarter: Improved simple methods for multiattribute utility measurement*, pages 306 – 325. Number 60. 1994.
- [6] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. In *Proceedings 15th International Conference on Machine Learning*, 1998.
- [7] H. In, D. Olson, and T. Rodgers. Multi-Criteria Preference Analysis for Systematic Requirements Negotiation. In *26th Annual International Computer Software and Applications Conference*, Oxford, England, August 2002.
- [8] J. Karlsson. Software requirements prioritizing. In *ICRE'96*, 1996.
- [9] F. Moisiadis. The fundamentals of prioritising requirements. In *System Engineering, Test and Evaluation Conference*, Sydney, Australia, 2002.
- [10] A. Ngo-The and G. Ruhe. Requirements Negotiation under Incompleteness and Uncertainty. In *Software Engineering Knowledge Engineering 2003 (SEKE 2003)*, San Francisco, CA, USA, July 2003.
- [11] B. Roy and D. Bouyssou. *Aide Multicritère à la Decision: Methods et Cas*. Economica, Paris, 1993.
- [12] G. Ruhe, A. Eberlein, and D. Pfahl. Quantitative winwin - a quantitative method for decision support in requirements negotiation. In *Proceedings 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02)*, pages 159 – 166, Ischia, Italy, July 2002.
- [13] T. L. Saaty. Fundamentals of the analytic network process. In *Proceedings of International Symposium on Analytical Hierarchy Process*, 1999.
- [14] A. Sivzattian and B. Nuseibeh. Linking the selection of requirements to market value: A portfolio-based approach. In *REFS 2001*, 2001.
- [15] P. Zave. Classification of research efforts in requirements engineering. *ACM Computing Surveys*, 29(4):315–321, 1997.

Team Tacit Knowledge as a Predictor of Performance in Software Development Teams

Sharon Ryan¹ and Rory V O'Connor²

¹*Business School, Dublin City University, Ireland,
sharon.ryan@dcu.ie*

²*School of Computing, Dublin City University, Ireland
roconnor@computing.dcu.ie*

Abstract. Tacit knowledge has been hailed as an important factor in team performance. In this paper we examine tacit knowledge as a significant predictor of team performance. This paper presents the results of a study of 48 software development teams where the influence of team tacit knowledge on team performance was examined. We also propose that team tacit knowledge is acquired through informal social interaction. In addition, reliability and validity for the team tacit knowledge measure in software development teams are established with respect to gut instinct, explicit job knowledge and experience.

1. Introduction

Tacit knowledge has been linked to team performance, in that teams with more tacit knowledge, are thought to be efficient and effective relative to other teams that members have known. In addition, tacit knowledge has been hypothetically connected to informal social interaction [20]. However there is little research on the measurement of tacit knowledge, with most research focussing on the measurement of individual tacit knowledge, while quantified field measurements of the quality social interaction are rare. The aim of this research is to examine the relationships between the quality of social interaction, tacit knowledge and team performance in software development teams. In addition, a measure of team tacit knowledge is developed and validated for software development teams.

1.1. The Research Context

Software development teams work with intangible cognitive processes rather than physical tangibles therefore the rules for developing tangible goods do not apply [4]. Members of software development teams are considered to be knowledge workers who are characterised as individuals who have high levels of

education and specialist skills combined with the ability to apply these skills to identify and solve problems. They also own the organisation's means of production (i.e. knowledge) [9]. The failure of many large software projects has highlighted the challenges in managing team-based knowledge work [10]. The majority of software projects do not meet budget and schedule, function unsatisfactorily and around 25% are never completed [12]. According to Brooks [4] 'there is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity'. However, research has shown that factors affecting team performance may be the key [14, 25].

Team performance on software development projects is dependent on many different and interacting factors e.g. effective plans, good communication, clear goals. In addition, internal group processes, particularly those focussing on the team's relationships, are more likely than technical factors to be associated with team performance on successful projects [14, 25]. In this research we focus on the importance of social interaction in acquiring knowledge, specifically tacit knowledge and impact of such knowledge on subsequent team performance.

1.2. Characteristics of Tacit Knowledge

Tacit knowledge, is not data or information, and cannot be codified. According to Polanyi [23] who coined the phrase; tacit knowledge is inarticulable and conceived through action. Tacit knowledge is most often conceptualised as distinct from explicit knowledge, but this distinction can be too simplistic, 'these two are not sharply divided. While tacit knowledge can be possessed by itself, explicit knowledge must rely on being tacitly understood and applied. Hence all knowledge is either tacit or rooted in tacit knowledge' [24].

There are two types of tacit knowledge: individual tacit knowledge and team tacit knowledge. Individual tacit knowledge is closely related to skills [24], team tacit

knowledge refers to knowledge associated with group activities, gained through the experience of working together by constructing shared cognitions or a 'collective mind' [3, 31].

Research on expertise indicates that much knowledge associated with successful performance is tacit [13] and this distinguishes more practically successful individuals from less practically successful [27]. The term tacit knowledge has in common parlance been associated with 'intuition' and 'gut instinct' [15]. Polanyi's conception of tacit knowledge has implications for its measurement, when we measure tacit knowledge we are really measuring 'implicit' knowledge which is articulable but previously unarticulated tacit knowledge [5]. Therefore, the term articulable tacit knowledge (aTK) [5] which refers to implicit managerial knowledge about software development projects, will be used as the definition for tacit knowledge.

1.3. Tacit Knowledge and Team Performance

Tacit knowledge has become popular in the management literature and is seen as a source of competitive advantage [3, 26] with some advocating the 'capture' of such knowledge [7, 20]. There are, however, only a few empirical studies of tacit knowledge and performance [14, 25] and in relation to tacit knowledge and team performance only one relevant empirical study exists [3]. In that study the effects of tacit knowledge on team performance were retrospectively investigated by examining two seasons of 23 professional basketball teams. The study concluded that team success increased as the team's tacit knowledge increased and concluded that tacit knowledge is gained through experience rather than formal study methods and can be acquired at an individual or group level.

1.4. How do we acquire tacit knowledge?

If we acknowledge, in the workplace at least, that 'the tacit dimensions of individual knowledge are not publicly available except as embodied in people to be hired' [21], then we must also recognise that capturing tacit knowledge inside the heads of people is impossible. Social interactions between people may be the route through which we acquire tacit knowledge, in that new knowledge is thought to be created through iterative social interaction [22], but not as first advocated, by making tacit knowledge explicit. Instead, a better explanation may be that 'new knowledge comes about not when the tacit becomes explicit, but when our skilled performance, our praxis-is punctuated in new ways through social interaction' [29].

Tacit knowledge originates with individuals and becomes group and organisational knowledge as a result

of social interaction [3]. Social interaction in groups is related to shared mental models, where team members tend to rely on one another in a cognitively interdependent manner.

Informal interpersonal communications are considered to be the principal way in which information flows into and through R&D organisations [1], with face-to-face interaction considered the richest medium for transferring knowledge because it allows for immediate feedback and the embodiment of tacit knowledge cues [19]. The goal of much face-to-face interaction is to disseminate information and pool diverse knowledge to make informed decisions [26]. The current definition of social interaction is face-to-face conversation, work related, personal or social that is informal. The interaction should not refer to formal interactions like a scheduled project meeting performance appraisal etc.

1.5. Measurement

Many theorists discuss tacit knowledge and its association with social interaction, but few measure it [7, 22]. Sternberg and others [16, 27] developed a practical approach to measuring tacit knowledge using a type of situational judgement test, where individuals are presented with a problem relevant to their domain followed by a set of options for solving a problem. Another included a social network analysis to examine formal and informal interactions in an IT department and concluded overwhelmingly that tacit knowledge is diffused in human to human interaction [5].

Each tacit knowledge measure must be created for its domain and context. Individual measures have been designed specifically for managers, military leaders [16, 27] and members of an IT department [5]. Team level tacit knowledge has been measured by proxy involving a retrospective analysis of the existing performance statistics to ascertain a measure of shared team experience [3].

In this study, tacit knowledge is measured at the team level. The rationale for the team level approach to tacit knowledge draws on the assumption that tacit knowledge is increased through social interaction and because people are different they will acquire different types and amounts of tacit knowledge which can be co-ordinated at a team level.

2. Tacit Knowledge and Performance Study

The aim of this study was to investigate the influence of team tacit knowledge on team performance in software development teams. In addition, reliability and validity for the team tacit knowledge measure were established, with particular reference to the importance social interaction in developing team tacit knowledge.

Forty eight teams from 46 small to medium sized organisations based in Ireland and the UK participated in an online interactive questionnaire. Team size varied from 2 to 12+, with the mean team size being 4.86 and an average within team response rate of 81.86%.

The first section of the questionnaire detailed the study and ensured anonymity. Completion of the questionnaire took approximately 15 minutes and included a measure of Team Tacit Knowledge (TTKM) designed specifically for software development teams, and index of the quality of social interaction within the team (QSI), two items measuring explicit knowledge and one item assessing 'gut instinct'. In addition experience in the software industry was gauged by tenure. These measures are detailed in the following sections.

2.1. Team Tacit Knowledge Measure (TTKM)

The rationale behind our measure of articulable tacit knowledge is based on the notion that experts differ from novices in the amount of tacit knowledge they possess. Items for the TTKM were elicited from 13 proficient or expert project managers using the repertory grid technique [25]. These items formed a 25 item bipolar questionnaire assessing articulable tacit knowledge about factors affecting team performance on successful software development projects.

This questionnaire was distributed to 18 experts and 124 novices. The 14 bipolar items that differentiated the two groups formed the present measure of articulable tacit knowledge. The 14 items are answered on a 5 point semantic differential type scale. An example of one of the bipolar constructs is "Innovative project <----> Mundane/Everyday type project." Respondents rated the constructs by selecting closest to the statement pole they felt described the factors that influence team performance on successful projects.

The tacit knowledge measure was scored by comparing the individual score on each of the 14 items with an expert profile. The average inter-rater reliability of the experts was 97.5% agreement. From their responses we constructed expert profiles using the expert mean. We scored the responses by calculating a squared Euclidean distance of the individual from that of the expert mean. These individual scores were then aggregated to form a team score. The average within team agreement was 95%.

2.2. Quality of Social Interaction

The Quality of Social Interaction (QSI) was assessed by a self-report questionnaire regarding two perceived outcomes of social interactions across team members, resulting in an index of social interaction. This measure was adapted from Chiu et al. [6] in which participants were asked to recall the most recent instance where they

spent more than 15 minutes alone interacting face-to-face with each member of the team. The two perceived outcomes referred to whether the interactions fostered (a) attainment of personal goals and (b) promoted positive feelings among participants. For each of the social situations participants were asked (a) to indicate on a 3 point, likert-type scale whether they had attained their goal in the interaction, and (b) indicate the degree of change in their relationship with the other person after the interaction, also on a 3 point scale¹.

In line with Chiu et al.'s analysis [6] for each interaction, the responses to these two questions were multiplied to form an interaction quality index for that social interaction. All of the interaction quality indexes were averaged to form an overall index of perceived interaction quality for each individual. These scores were then aggregated to form a team score of social interaction.

2.3. Team Performance

Two dimensions of performance for knowledge teams consisting of effectiveness and efficiency were measured [10]. Objective measures of performance present difficulties in the IS field [17], since 'using objective measures assumes comparability across software projects or unique situations constraints, and this raises a new set of methodological measurement issues' [12].

The Effectiveness measure constituted 5 items and asked how well teams performed, in relation to other software development teams they have known, on dimensions of work quality, team operations, ability to meet project goals, extent of meeting design objectives and reputation of work excellence. The Efficiency measure had two items and dealt with adherence to schedule and budget. Responses for both effectiveness and efficiency were rated on a 1 to 5 likert-type scale from 'not very good' to 'excellent'.

2.4. Explicit Knowledge and Gut Instinct

Two self report items measured perceived explicit knowledge which was operationalised as official job knowledge. Explicit knowledge was assessed by asking respondents their levels of familiarity with official written procedures and their degree of reliance on official written procedures involved in carrying out their work. In addition team members were asked the extent to which they rely on their gut instinct in doing their job. Gut instinct was defined as implicit subjective procedures and

¹ Likert scales are very commonly used with interval procedures in the social sciences. In a review of the literature Jaccard and Wan [18] found that "for many statistical tests, rather severe departures (from intervalness) do not seem to affect Type I and Type II errors dramatically."

standards that are difficult to articulate but can be seen in practice. All items were scored on scale from 1 to 5.

2.5. Overall Scoring, Reliability and Validity of Tacit Knowledge Measure

First individual scores were calculated for all variables. Then these were averaged for team level analysis. A preliminary validation of the tacit knowledge measure was undertaken to evaluate the discriminant validity of the Team Tacit Knowledge Measure (TTKM) relative to explicit knowledge; convergent validity in relation to years of experience, gut instinct, social interaction and predictive validity in relation to team performance. We expect that scores on the TTKM would be unrelated to explicit job knowledge and that teams with more years of experience and more reliance on gut instinct and higher social interaction would possess more tacit knowledge. It is also expected that scores on the TTKM would predict team performance over and above all other factors.

3. Results and Analysis

First we examined the reliability of the TTKM at the individual and team level. Tacit knowledge inventories and other situational judgement tests differ from conventional knowledge tests in that items may be poorly defined and are multidimensional in nature drawing on skills, knowledge and abilities [16]. Across an inventory there are diverse areas of knowledge some acquired by the individual some not, therefore the complexities of the tacit knowledge measures reduces the likelihood of obtaining the same levels of internal consistency as for other traditional knowledge and ability tests. According to Legree expect to obtain alpha coefficients between .5 and .8 [20].

Internal consistency for the TTKM as measured by Cronbach's coefficient alpha was $\alpha=.493$ at the individual level and $\alpha=.710$ at the team level. Indicating a significant increase in the internal reliability of the measure at the team level, thus providing support for the premise that TTKM measures tacit knowledge at the team rather than individual level. Given that the obtained team level reliability falls within the range for other situational judgement tests and for those reliabilities obtained on previous measures of tacit knowledge [16] then we consider the internal consistency of the team level score to be acceptable.

3.1. Relationships among predictors

Initially we first computed a correlational analyses to assess the extent of discriminant and convergent validity

between the tacit knowledge measure and other predictors. This data is provide in table 2 of appendix A.

We first explore the validity of the TTKM. As predicted, tacit knowledge was not related explicitly to knowledge as measured by 'reliance on written procedures' and 'familiarity with written procedures', thus providing divergent validity for the TTKM. The TTKM was not significantly related to gut instinct perhaps because gut instinct may be seen as comprising two aspects: gut decisions and gut reactions.

Gut decisions are 'based on instinct and experience tempered by information and a broad sampling of opinions. Gut reactions conversely are 'based on instinct that's overwhelmed by a compelling piece of information or by the heat of the moment' [2]. Gut decisions are based therefore on information and experience, and this is borne out in the significant association between the experience and gut instinct ($r=.459$, $p<.01$). Gut instinct appears to be an individual level variable since it is also unrelated to quality of social interaction and therefore is not related to team tacit knowledge. In addition, a single item measure, such as this, would lack the scope to encompass all the aspects of gut instinct.

The TTKM did not correlate as expected with experience as measured by years in the software development industry, this may be because time is considered important to the development of knowledge and skill, it is not necessarily an indicator of the amount of development that has occurred [16]. In software development, deliberative reflection rather than tenure may be the key as to why people differ in the amount of knowledge gained, since experts tend to engage in deliberate and reflective practice [30].

Convergent validity was provided by a significant correlation between scores on the TTKM and quality of social interaction ($r=.450$, $p<.01$) providing empirical support for the theoretical argument, that tacit knowledge is diffused and acquired through social interaction. In terms of predictive validity TTK was significantly related to the effectiveness component of team performance ($r = .345$, $p<.05$) but not the efficiency aspect, with effectiveness and efficiency correlating well together ($r = .547$, $p<.01$). This finding is consistent with the nature of efficiency and effectiveness in teams.

Efficiency relates to budgeting and scheduling and has been found to be associated with formal administrative co-ordination and reporting procedures which themselves have not been found to be related significantly to effectiveness [10]. Effectiveness on the other hand is characterised by how well the team meets project goals, the quality aspect rather than speed and budget.

3.2. Other Correlations

The two items measuring explicit knowledge were significantly related to one another ($r=.473$, $p<.01$). Reliance on gut instinct and 'familiarity with written procedures' were both significantly related to effectiveness ($r=.302$, $p<.05$; $r=.296$, $p<.05$, respectively). These findings suggest that teams who were more familiar with written procedures, are more effective but not so if they rely on these procedures, however reliance gut instinct increases effectiveness. The only significant relationship to efficiency was 'familiarity with written procedures' ($r = p<.05$), suggesting that this aspect of explicit knowledge is an indicator of overall team performance.

3.3. Tacit knowledge as a Predictor of Team Performance

Scores on TTK were correlated significantly with the team performance measure of effectiveness but not efficiency. We conducted a hierarchical regression in order to ascertain the extent to which tacit knowledge in software development teams accounts for unique variance in effectiveness ratings. In the hierarchical regression we entered QSI, Experience, Gut Instinct, Familiarity with written procedures and reliance on written procedures as control variables in step 1. Scores on the TTKM were entered in step 2.

The results illustrated in table 1 indicate that around 27.8 % of the variance in effectiveness is accounted for by the all of the variables combined. The control variables explain 18.2% of the variance in effectiveness and team tacit knowledge describes 9.6% of variance in effectiveness over and above all other factors in this study ($p<.05$). Therefore, team tacit knowledge is a significant predictor of effectiveness.

Table 1. Hierarchical regression for team effectiveness

Independent variables	Standardised beta weights	
	Step 1	Step 2
Step 1: Controls		
QSI	-.028	-.188
Experience	.065	.079
Gut	.271	.292
Familiarity	.247	.219
Written	.067	.038
Step 2: Team tacit knowledge		
TTKM		.352*
R^2	.182	.278
ΔR^2		.096*
ΔF		5.454*

* $P<.05$

4. Conclusions

The results suggest that tacit knowledge plays a significant role in explaining team effectiveness but not efficiency and that scores on the TTKM are a significant predictor of team effectiveness over and above all other factors in this study, signifying its importance in software development teams.

A limitation of this study is that there is no way of knowing if the teams collaborated or interacted with one another while completing the questionnaire. However, the existence of standard deviations across responses, on all measures in all teams provides some support that the teams did not collaborate.

The type of tacit knowledge related to social interaction is team based, and involves interactions between team members who share and acquire this knowledge, with different team members possessing different aspects of the team tacit knowledge. The implications for software teams is that since tacit knowledge leads to more effective teams, and team tacit knowledge is acquired through social interaction, then it is important to encourage informal social interaction to increase team level tacit knowledge.

We cannot draw firm conclusions as to how managers go about increasing social interaction as it was not addressed in this study. However, suggestions forwarded by DeMarco and Lister [8] regarding the arrangement of office space in order to balance privacy and informal interactions in the workplace would be useful. Also, research in the realm of ecological psychology illustrates how the design of our workplace affects our social interaction [11].

4.1. Continuing Research

This study is part of a larger study which explores the quality and quantity of social interaction in software development teams and their effect on the acquisition and transfer of articulable tacit knowledge. Mediating variables such as transactive memory (social cognition), climate and knowledge sharing practices are also investigated. Team performance and product performance form the dependent variables of the larger study.

References

1. T. Allen, "Managing the Flow of Technology", MIT Press, Cambridge, Mass., 1977.
2. Anon., "Protect Yourself from too Much Intuition", New Zealand Management, Vol. 45, No. 7, 1998, p. 24.
3. S. Berman, J. Down and C. Hill, "Tacit Knowledge as a Source of Competitive Advantage in the National Basketball Association," Academy of Management Journal, 2002, Vol. 45, No. 1, 13-31.

4. F. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering", *Computer*, Vol. 20, No.4, 1987
5. P. Busch, D. Richards and C.N.G. Dampney, "The Graphical Interpretation of Plausible Tacit Knowledge Flows", "The Graphical Interpretation of Plausible Tacit Knowledge Flows", Vol. 24, 2003.
6. C. Chiu, Y. Hong, W. Mischel, Y. Shoda, "Discriminative Facility in Social Competence: Conditional Versus Dispositional Encoding and Monitoring-blunting of Information", *Social Cognition*, Vol. 13, No. 1, 1998.
7. S. Droege and J. Hoobler, "Employee Turnover and Tacit Knowledge Diffusion: A Network Perspective", *Journal of Managerial Issues*, Vol. 15, No.1, 2003, pp.50-64.
8. T. DeMarco and T. Lister, "Peopleware: Productive Projects and Teams" 2nd Edn. Dorset House, NY 1999.
9. P. Drucker, "Post-Capitalist Society", Harper Business, New York, 1993.
10. S. Faraj and L. Sproull, "Coordinating Expertise in Software Development Teams", *Management Science*, Vol. 9, No. 2, 2000, pp. 1554-1568.
11. W.Gaver, "Affordances for Interaction" *Ecological Psychology*, Vol.8, No. 2, 1996, pp. 111-129.
12. W. Gibbs, "Software's Chronic Crisis", *Scientific American*, September, 1994, pp. 86-95.
13. R. Glaser, M.T.H.Chi, and M.J. Farr, "The Nature of Expertise" Erlbaum, Hillsdale, New Jersey, 1988.
14. P. Guinan, J. Coopridge, and S. Faraj, "Enabling Software Development Team Performance During Requirements Definition: A Behavioural Versus Technical Approach", *Information Systems Research*, Vol. 9, No. 2, 1998.
15. A. Hayashi, "When to Trust your Gut", *Harvard Business Review*, Vol. 29, No. 2, 2000, pp. 59-66.
16. J. Hedlund, G. Forsythe, J. Horvath, W. Williams, S. Snook and R. Sternberg, "Identifying and Assessing Tacit Knowledge: Understanding the Practical Intelligence of Military Leaders", *The Leadership Quarterly*, Vol. 14, 2003, 117-140.
17. J. Henderson, and S. Lee, "Managing I/S Design Teams: A Control Theories Perspective", *Management Science*, Vol. 38, No. 6, 1992, 757-777.
18. J. Jaccard, and C.K. Wan (1996). *LISREL approaches to interaction effects in multiple regression*. Thousand Oaks, CA: Sage Publications.
19. K. Koskinen, P. Pihlanto, H. Vanharanta, "Tacit Knowledge Acquisition and Sharing in a Project Work Context", *International Journal of Project Management*, Vol. 21, 2003, 281-290.
20. P. Legree, "Evidence for an Oblique Social Intelligence Factor Established with Likert-based Testing Procedure", *Intelligence*, Vol. 21, 1995, 247-266.
21. D. Leonard and S. Sensiper, "The Role of Tacit Knowledge in Group Innovation", *California Management Review*, Vol. 40, No. 3, 1998, pp.112-132.
22. I. Nonaka, and H. Takeuchi, "The Knowledge Creating Company", Oxford University Press, Oxford, 1995.
23. M. Polanyi, "The Tacit Dimension", Peter Smith, 1966.
24. M. Polanyi, "Knowing and Being", Routledge, 1969.
25. S. Ryan and R. O'Connor, "The Elicitation of Expert Knowledge regarding the Factors that affect Team Performance on Software Development Projects", in 16th International Conference on Software & Systems Engineering and their Applications, Paris, 2003
26. G. Stasser, "Pooling of Unshared Information During Group Discussion", in S. Worchel, W. Wood, & J.A. Simpson (eds.), *Group Process and Productivity*, Sage, Newbury Park, CA, 1992, pp.48-67.
27. R.J. Sternberg, G.B. Forsythe, J. Hedlund, J. Horvath, R.K. Wagner, W.M. Williams, S.A. Snook, and E.L. Grigorenko, "Practical Intelligence in Everyday Life", Cambridge University Press: Cambridge, 2000.
28. D.J. Teece, and G. Pisano, "The Dynamic Capabilities of Firms", In G. Dosi et al. (Eds.), *Technology, Organization and Competitiveness*, pp. 193-212. Oxford University Press, Oxford, 1998.
29. H. Tsoukas, "Do We Really Understand Tacit Knowledge?", in M. Easterby-Smith and M. A. Lyles (eds.), *Handbook of Organizational Learning and Knowledge*, Blackwell, Oxford, forthcoming.
30. I. Vessey, "Expertise in Debugging Computer Programs: A Process Analysis", *International Journal of Man-Machine Studies*, Vol. 23, No. 5, 1985, pp. 459--494.
31. K.E. Weick, and K.H. Roberts, "Collective Mind in Organizations: Heedful Interrelating on Flight Decks", *Administrative Science Quarterly*, Vol. 38, 1993.

Appendix A

Table 2. Means, standard deviations and intercorrelations for teams

n = 48	M	SD	1	2	3	4	5	6	7
1. TTKM	10.84	2.48	-						
2. QSI	12.83	1.90	.450**	-					
3. Experience	11.99	6.04	.202	.510**	-				
4. Written	3.00	0.95	.204	.204	.238	-			
5. Familiarity	4.07	0.76	.183	.177	.259	.473**	-		
6. Gut	1.99	0.62	.042	.259	.492**	.017	.079	-	
7. Effectiveness	18.43	2.76	.345*	.134	.261	.201	.296*	.302*	-
8. Efficiency	6.48	1.48	.089	-.035	.014	.158	.329*	.098	.547**

*p<.05, **p<.01

Towards Effectively Appraising Online Stores

Dr Ernest Cachia, Mark Micallef

Department of Computer Science and Artificial Intelligence

University of Malta

ernest.cachia@um.edu.mt, mmic048@um.edu.mt

Abstract. This paper introduces research being carried out into the measurement of the quality of e-commerce systems. Considerable work has been done on software metrics in the last few decades but e-commerce specific metrics seem only applicable to already deployed systems. It is proposed that a set of metrics is needed, which can be applied from the earlier stages of e-Commerce system development to improve risk management. This paper attempts to appraise e-commerce systems by proposing a set of essential attributes for an e-commerce site to succeed. This paper also serves as groundwork for future e-commerce metrication work based on these same attributes.

Keywords: E-Commerce, Survey, Software Quality Assurance, Software Measurement, Security.

1.Introduction

Electronic Commerce (e-commerce) is most often referred to as *the buying and selling of products and services using the Internet*. The British government broadly and completely defines e-commerce as *“the exchange of information across electronic networks, at any stage in the supply chain, whether within an organisation, between businesses, between businesses and consumers, or between the public and private sectors, whether paid or unpaid”* [1]. Throughout this paper, references to e-commerce systems should be taken to imply a Business-to-Consumer (B2C) type model.

Whatever definition one gives it, e-Commerce is fast becoming a popular means of purchasing almost anything you need from books to diamonds. So much so that it has become an almost discounted fact for a modern-day business to provide its goods/services online. Research reveals that in 2003 online retail sales in the US again jumped by 29.1% from \$13.8 billion in 2002 to \$17.8 billion in 2003. To put things in perspective, in 1999 (only) \$5.4 billion were spent online in the US [2]. Similar trends have been observed in Europe.

With this in mind, focus is naturally drawn to the quality of IT systems used to facilitate commercial transactions, and more specifically the quality of e-commerce systems. The issue of classifying e-commerce systems as being of definable quality can be approached from two aspects: the technical and the business aspects. The technical aspect deals with how such systems actually work. The business aspect is more related to products/service handling. Bearing in mind that an e-commerce initiative (venture) is made up of a sound business model and technical innovation, both technical and business aspects of e-commerce must go hand-in-hand in order for an e-commerce venture to succeed. Although both the technical and business aspects are important for the success of an e-commerce venture, this paper will focus on technical issues.

In order to substantiate this argument, results of various research exercises have been utilised, including a survey which was carried out by the authors of this paper amongst 350 regular e-commerce users.

2.How is E-Commerce System Measurement Different?

Apart from the six generic software quality attributes as set out by ISO-9126 [6], e-Commerce systems contain certain attributes which would seem to feature more strongly in them than in more generic software systems [10].

Since the first software metrics appeared in the 1970's, new ones have been developed as new technologies surfaced (e.g. Childamber and Kemerer's metrics suite for object oriented design [9]). It is the opinion of this paper's authors, that e-commerce systems have sufficient unique characteristics to merit their own e-commerce metrics and measurements suite. This is not to say that “conventional” attributes need not be present in e-commerce systems. On the contrary, they maintain their fundamental importance. Related work has been carried out in this area by Stefani [20] and Barnes [21].

The authors of this paper are of the opinion that e-commerce applications are a subset of web applications which in turn are a subset of generic software applications. Furthermore, many core aspects and functionality of e-Commerce systems can be viewed as a specialised class of web applications. It is therefore predictable that fundamental web application characteristics are also found within e-Commerce systems. This is supported by the distinguishing features of e-Commerce systems as highlighted in bold text below...

First of all, e-commerce systems are largely **content-driven**. Customers log on looking for information about products/services, be it a simple price and description, a very detailed technical specification, or the terms of purchase. Enabling customers to effectively browse through a myriad of products and providing them with exactly all the information they need can be a challenging task, especially when considering that different customers may have different needs. Issues arise as to how to organize content in a system, how to prioritize it, how to allow users to navigate through content and so on. Clearly, **navigability** would be an important attribute to consider when appraising an e-commerce site.

More than any other type of software applications, e-commerce systems are **exposed to the world**. Given the open nature and main intent of the Internet many aspects of the Internet can work against the security interests of an e-commerce website [10]. This highlights two other important attributes: security and privacy. In most cases, a customer will trust an online vendor with personal information with the proviso that his/her personal data is protected and will not be misused in anyway. A betrayal of this trust, whether intentional or not, could have serious negative repercussions on the vendor.

Another distinguishable feature of e-commerce systems is that they are mostly **browser-based**. The HTTP protocol is not so user-friendly when compared to the event-driven programming that most of us are used to when developing user-interfaces. Functionality that we have grown accustomed to in other paradigms present a much tougher challenge. Scripting languages such as JavaScriptTM and more recently the emergence of the Microsoft .NetTM framework attempt to tackle this problem but then again, support for these technologies is not the same on all browsers [11]. This presents problems relating to usability and portability of web systems.

A site becoming popular will generally translate to an increase in profits to a vendor but there is another side to having an **enormous user base**. It should be ensured, that

the site performs as well with a million hits a day as it would with 1000 hits a day. As Deters [11] puts it, "having fast and dependable access to the most relevant information available is of the utmost importance in a competitive information-oriented society". Therefore, performance and scalability become key issues. The research presented later in this paper indicates that only 18% of users are likely to remain unconditionally loyal to an e-commerce site after its performance degrades due to increased popularity. Another problem associated with having a large number of hits is the problem of portability. The higher the number of hits experienced by an e-commerce site, the higher the chances are that the given site is being accessed from different devices, operating systems, and browsers. This can cause problems especially if a site is using technologies that are not universally implemented or uniformly rendered in different environments. Also, having a large customer-base poses a problem with defining a mechanism for customer feedback.

Lastly, e-commerce systems are **likely to change quite often**. Whether it be changing site content, site aesthetics or even site functionality. Just like in any software system, changes to a site will introduce additional risks of failure thus affecting its reliability. Clearly, a change in site functionality carries more risk than a change in content. However, even a simple change in website content can bring with it layout problems (text too long, image of an incorrect size or missing, etc.) potentially causing a deterrent to new customers. Developers should make sure that a site ages well, indeed matures, as changes are implemented. This is reflected in the generic software attribute of maintainability.

3.Survey Design

A survey "can be a powerful tool to figure out what your market needs and how you can market to them" [18]. The main raison d'être for online stores is to be "sold" so-to-speak to everyday online shoppers. Therefore it was deemed imperative at this stage to elicit and highlight online shopper opinion.

On the basis of the e-commerce characteristics identified in section 2, a set of related quality attributes was derived and a survey was designed to help obtain a user perspective on the issues involved in e-commerce systems appraisal. The survey was divided into two sections. The first section focused on collecting information about the participants that would later help filter results and identify sub-trends according to certain criteria (e.g. age, education level, etc). The second section was designed to

“tap” into the participants’ views on the quality of e-commerce systems.

Based on the discussion in section 2 of this paper, the following attributes were felt to be relevant regarding e-commerce systems:

- Security and Privacy
- Portability
- Performance and Scalability
- Navigability, Usability and Aesthetic Features
- Multi-lingual Features
- Low-Bandwidth version of sites
- Reliability

The questionnaire was therefore designed to elicit information relevant to the above attributes. It should be noted, that due to paper length requirements, it was decided not to include the explanation and justification regarding the structure and choice of questions in the survey. However the actual questionnaire together with supporting explanatory documentation can be accessed at <http://www.cs.um.edu.mt/~mmica/survey>

4.Results

This section will discuss results from the survey and propose a set of attributes which should be deemed essential in an e-commerce system in order for it to be considered a quality system.

Please note, that some figures collectively amount to more than 100% because users were offered the possibility to make multiple choices.

4.1.General Observations

One of the first steps taken when analyzing the data was to analyze the user sample. Just over 95% of participants claim to use Internet Explorer™ whilst 7% use Netscape™. Other browsers compare poorly. Also, the Microsoft Windows™ OS family seems to be the most popular amongst our sample with over 98% of users using Windows™. Linux/Unix come in second with 8.92%. With regards to device usage, the desktop PC claims the top spot with 92.9% of users using desktop PCs for e-commerce transactions. 24% use laptops whilst mobiles and PDAs trail with 5.23% and 1.85% respectively. These figures compare well with other usage

surveys that have been carried out [12] [13].

Regarding demographics, 94% of participants were under 50 years old and the most popular items bought online are books with 80% of users having bought books online. Consequently, other popular purchases ranked as follows: Software (32%), hardware (29.6%) and music (29.3%). 71.9% of users claim they would be negatively affected had e-commerce not been available to them.

Disturbingly, 77% of users claim to have abandoned transactions mid-way through. The top reasons for this were stated as:

- User decided (s)he did not want the product (43%)
- Website Error (36%)
- Purchasing Process too long (35%)
- Site too slow (33%)
- Delivery, Payment, or pricing problems (14%)
- Browser Compatibility Problems (4%)

4.2.Security and Privacy Issues

Security turned out to be the attribute that was perceived by most participants to be of great importance. 35% of respondents claimed that if they were to choose a reason not to use e-commerce, it would be for fear of compromised security. It is interesting to note that another 30% would not choose e-commerce because they prefer to physically touch goods before buying them. This might be interpreted as users not trusting online vendors outright when it comes to delivering good quality products. Also, when asked how sure they would have to be of a site’s capability to offer them security and privacy before they purchased from it, 43.5% of the users said they would have to be as certain as they possibly can be (water-tight privacy policy, secure connection, etc.). A further 42% said they would also buy if they had minor doubts (such as there being a risk of the site giving their e-mail to third parties). Security was placed first when participants were asked to rank quality attributes in order of importance. It obtained an average score of 6.235 (out of a maximum of 7). Surprisingly, 33 participants 13.87% claimed that security was not an essential attribute they would look for in an online store. However, on closer examination, these

participants might have been inconsistent because when looking at their answers in isolation, they still placed security as the most important attribute with a score of 6.182. It can be safely concluded that security is an indispensable attribute in e-commerce systems.

4.3.Portability

Portability in e-commerce systems refers to the extent to which a system is accessible from different operating systems, browsers and devices without loss in functionality. The case for portability is not a strong one if one relies on the results of this survey. Consider the following results:

1. Participants ranked portability as the 5th most important attribute (out of 7)
2. 98% of participants use WindowsTM-based OSs
3. Almost 93% of participants use Desktop PCs
4. Over 95% of participants use Internet ExplorerTM
5. Less than 4% of users who abandoned transactions midway through did so because of compatibility problems. Less than half of these users were using Internet ExplorerTM

The results seem to suggest that if an e-commerce system were to be tailored for WindowsTM-based PCs or laptops using Internet ExplorerTM, any portability problems with other systems would cause minimal negative repercussions in the vendor's business.

Nevertheless, one should always remain vigilant with regards to portability issues. Firstly, when asked whether or not they would be willing to install another browser if an e-commerce site was not compatible with the one they were currently using, 88.65% of users said they would not. Therefore one must keep a close eye on the market share commanded by browsers and operating systems and invest in the portability of e-commerce sites as necessary. Another concern is the much talked about rise of mobile commerce (m-commerce). Even though early high hopes for M-Commerce failed to materialise in the first years of this century [15], falling costs and faster mobile networks have raised hopes on the feasibility of M-Commerce [16].

Portability is being recommended as a necessary attribute by the authors of this paper although in the current

environment, compatibility with dominant technologies would seem to ensure a far greater reach and influence of online shoppers.

4.4.Performance and Scalability

Speed is important to users. Over 33% of users who abandoned transactions mid-way through did so because the site was too slow. Also, when asked how they would react if their favorite e-commerce site experienced performance degradation due to popularity, only 18.4% of users claimed they would remain loyal. However, 57% claimed they would try to use the site at off-peak times in order to get better performance. It is also worth noting, that 22% of participants consider the speed of a site the most important "first impression" factor. This would mean that when they first visit a site, the primary deciding factor on whether they decide to stay or not is performance. This is backed up by the popular *7-second rule*, which states that a web page should take no more than 7 seconds to download and display to the site visitor on the slowest probable connection [10]. Participants rated performance as the 4th most important attribute in an e-commerce system with an average score of 4.128 (out of a possible 7). The authors are therefore recommending performance as a required attribute in all e-commerce systems.

4.5.Navigability, Usability and Aesthetic Features

Of the 77% of users who have abandoned transactions mid-way through, 35.6% did so because the process was seen as being too long. Also, these same users were more likely to look for an alternate site after abandoning a transaction rather than try again or contact the vendor by other means. This suggests that poor usability will have a tremendous impact on the success of an e-commerce site.

72% of users know beforehand what they are looking for when visiting an e-commerce site. This indicates that good search and navigation features are important in e-commerce systems. 30% of participants also chose good navigation as the primary *first impression* factor.

With regards to aesthetics, only 6.7% of users rate aesthetics as the primary *first impression* factor.

Navigability was ranked as the third (out of seven) most important attribute in e-commerce systems with an average score of 5.492 (out of a possible 7).

Usability with an emphasis on navigability would therefore be recommended as an essential attribute of e-commerce systems.

4.6. Multilingual Features

The importance of multilingual features depends very much on the context in which an online store is operating. For example, in some regions of northern Italy, both Italian and German are spoken to varying degrees. Therefore, an online store servicing such regions would do well to provide both Italian and German versions of its site. Respondents ranked multilinguality as the sixth (out of seven) most important attribute with an average score of 2.043 (out of a maximum of 7). Also, almost 51% of participants claim that an e-commerce system could still be classified as a quality e-commerce system if it did not have multilingual features. The reason for this might be that one tends to automatically use e-commerce sites in one's own language. Therefore users might not really be aware of the multilinguality problem.

Providing multilingual versions of an e-commerce site does not simply involve translating content into different languages. Besides there being a vast research area into the technical issues involved in offering such a service, there are also accompanying business and legal structures that would also need to be set up. For example, what happens if Japanese speaker places an order and needs customer support? The vendor would also need to have multilingual customer relationship management (CRM) processes in place. A recent study claimed that 46% of companies interviewed turn away international orders because they do not have the processes in place to handle them [19]. Implementing such processes would clearly require a certain amount of resources which most businesses, especially those in their early stages might not be able to afford. Therefore it might make more sense for businesses that are not large multi-national corporations to concentrate on markets where only one language is spoken and expand multilingual features as business grows.

Although a desirable attribute, multilinguality is not being

recommended as an essential attribute by the authors of this paper.

4.7. Low-Bandwidth Version of Site

Some e-commerce sites have a text-only or low-bandwidth version available for users with slow connections. When asked about the importance of this feature, participants in the survey ranked it as being the least important attribute of all (average score of 1.587). Also, 52.5% of participants deem the feature to be unnecessary in an e-commerce site. Considering the increased Internet bandwidth available to users and the increasing popularity of broadband communications [17] as well as the giant strides in technology as a whole, the authors are not recommending low-bandwidth versions as an essential attribute of an e-commerce system.

4.8. Reliability

Reliability is considered to be an extremely important attribute in e-commerce systems because of the following indicators:

1. 36.6% of users have abandoned transactions midway through due to website errors
2. Reliability was ranked as the second most important attribute by participants with an average score of 5.55 out of a possible 7.
3. Almost 37% of users consider a site's reputation as the primary *first impression* factor when they first visit it and a site with frequent errors and crashes is unlikely to gain a good reputation.

Considering the above results from the survey, the authors recommend reliability as an essential attribute of e-commerce systems.

5. Conclusions and Future Work

Based on results of the survey and other research cited in this paper, a number of quality attributes are being recommended as being essential to e-commerce systems. That is to say, that no e-commerce system can be reasonably expected to succeed if it does not exhibit a considerable degree of each recommended attribute. The

attributes are also being given an importance ranking as follows (most important first):

1. Security
2. Reliability
3. Navigability
4. Performance
5. Portability

Further work needs to be done before online store quality can be effectively measured. One still needs to define the minimum level of each characteristic that an e-commerce system needs to exhibit. In order to do that, each attribute should be measurable in some way. Therefore, the next step in the ongoing research will be to define a metrication and measurement framework for these attributes. When that is achieved, some form of progressive benchmarking system could be defined whereby e-commerce systems can be classified as being of a given quality depending on the level of each attribute exhibited by the system.

References

- [1] Prime Minister's Strategy, "E-Commerce@its.best.uk", www.number-10.gov.uk/su/ecommerce/body.pdf, 1999
- [2] Emarketer.com "Online Retail Update: Latest Q4 Quote", www.emarketer.com/news/article.php?1002631, Emarketer.com, 2004
- [3] Crosby P.B., "Quality is Free: The Art of Making Quality Certain", McGraw-Hill, 1979
- [4] Chaffey D., "E-Business and E-Commerce Management" Financial Times / Prentice Hall, 2002
- [5] Lee J., Podlaseck M., "Using a Starfield Visualization for Analyzing Product Performance of Online Stores", Proceedings of the 2nd ACM conference on Electronic commerce, 2000
- [6] "ISO/IEC 9126:2001 – Software Engineering Product Quality", International Standards Organization for Standardization, 2001
- [7] Kafura D. "A Survey of Software Metrics", Proceedings of the ACM annual general conference on the range of computing, 1985
- [8] McCabe T.H., "A Complexity Measure", IEEE Transactions on Software Engineering, 1976
- [9] Chidamber S.R., Kemerer C.F., "Towards a Metrics Suite for Object Oriented Design", ACM Conference proceedings on Object-oriented programming systems, languages, and applications, 1991
- [10] Dustin E. et al, "Quality Web Systems", Addison Wesley, 2001
- [11] Chandra K., Chandra S.S. "A Comparison VBScript, JavaScript and JScript", Journal of Computing in Small Colleges (2003)
- [12] Deters R., (2001) "Scalability and Information Agents", ACM SIGAPP Applied Computing Review
- [13] www.w3schools.com, "January 2004 browser usage statistics", http://www.w3schools.com/browsers/browsers_statistics.asp, 2004
- [14] www.arkom.co.uk - "Web Browser Usage Survey 2003", <http://www.arkom.co.uk/news-article.asp?NewsID=42>, 2003
- [15] Mahoney M, "Whatever happened to mobile commerce?", E-CommerceTimes.com, <http://www.ecommercetimes.com/perl/story/15042.html>, 2001
- [16] Halper M, "Back From the Dead", Time Europe Magazine Vol. 163 No.7, 2004
- [17] Gill L, "Study: Broadband Adoption on the Rise", E-CommerceTimes.com, <http://www.ecommercetimes.com/perl/story/18355.html>, 2002
- [18] TWM, "How to Design a Survey", www.thewritemarket.com/marketing/survey-design.htm, 2003
- [19] European Business Management School, "SME Management of Multilingual Web sites", <http://www.global-presence.org.uk>
- [20] Stefani A., Michalis X., "A Model for Assessing the Quality of E-Commerce Systems", Proceedings of the PC-HCI 2001 Conference on Human Computer Interaction, Patras, 2001
- [21] Barnes S.J., Vidgen R.T., "An Integrative Approach to the Assessment of E-Commerce Quality", University of Bath

UCDA: Use Case Driven Development Assistant Tool for Class Model Generation

Kalaivani Subramaniam, Dong Liu, Behrouz H. Far and Armin Eberlein
Department of Electrical and Computer Engineering, University of Calgary
2500, University Drive, N.W., Calgary, Alberta, Canada, T2N 1N4
{subrama, liud, far, eberlein}@enel.ucalgary.ca

Abstract. *The development of class models using the Rational Unified Process (RUP) requires complete, correct and unambiguous use case specification documents. The Use Case Driven Development Assistant (UCDA) tool provides automated assistance in developing use case diagrams, writing use case specification documents and developing the analysis class models. UCDA uses a freely available natural language parser and Rational Rose's extensibility interface to support the automation of the Object Model Creation Process (OMCP). The parser is a shift-reduce parser and is implemented in Python. This paper introduces the UCDA tool and its application in OMCP. The process of automation is illustrated in a case study of an Automated Teller Machine (ATM) System. The UCDA tool increases design productivity, reduces time-to-market and is of great help to novice software developers.*

1. Introduction

Object Oriented Analysis and Design (OOAD) is a software development paradigm widely used in software development. Identifying objects and classes from requirements [8] that are represented in natural language is an essential task in OOAD.

UCDA employs common requirements elicitation techniques to gather requirements and to document them in the requirements document. The software designer then analyzes the requirements and identifies the objects with the Object Model Creation Process (OMCP) [4]. Attributes, associations and behaviour of objects are also established as part of this model. Later, the object model is refined using generalization, and objects and classes are identified based on domain knowledge, real world experiences and user interviews. Tools that implement this process are already available [4].

NIBA (Natural Language Requirements Analysis) is an approach that starts with linguistic analysis and transforms a textual requirements specification into a conceptual predesign schema, which is then validated and mapped onto conceptual schema [1]. This tool parses the

requirements in German. The tool LInguistic Assistant for Domain Analysis (LIDA) processes text to develop object models. It analyzes the text to identify the model elements; then the model elements are refined through a validation process [2].

Our methodology follows the IBM Rational Unified Process (RUP) approach to automate the class model generation. "RUP is a configurable software development process platform that delivers proven best practices and a configurable architecture" [3]. RUP implements several best practices in software engineering. It specifies the functional behaviour of a system using use cases. Use case model development is a kind of knowledge elicitation.

The structure of this paper is as follows. Section 2 provides a methodology for developing the class model. Section 3 describes UCDA, a tool that is designed to generate the use case model, robustness diagrams, collaboration diagrams and class diagrams. Section 4 presents a case study of the proposed system. Finally, Section 5 provides conclusions.

2. Methodology

Our methodology requires a careful analysis of the stakeholders' requests, which are stated in textual form. Use cases and classes are then identified based on predefined rules. The method follows the Rational Unified Process (RUP) approach to develop the use case model and the class model. The models are developed using Rational Rose according to the Unified Modeling Language (UML) standards.

The process starts with stakeholders' requests of the proposed system. Typically these requests are parsed by the natural language parser and then analyzed by the system to identify actors and use cases. The relationships among actors and use cases are identified and the use case diagram is developed using these artifacts. Detailed information about each use case is collected from the stakeholders. This information is formalized according to a use case template, validated and collected in a use case

specification document. The document along with the use case diagram serves as the source of information for developing the class diagram. The objects are identified from the use case specification document and categorized into boundary objects, entity objects or control objects. The robustness diagrams are developed to show the different types of objects and their relationships to one another. Then the collaboration diagrams are generated to show the messages passed between the objects. A glossary is used during this process to prevent ambiguity and increase consistency. Finally objects are refined to generate the class diagram.

The activity diagram showing the workflow of the UCDA tool is shown in Fig.1. An example of using this methodology is given in Section 4.

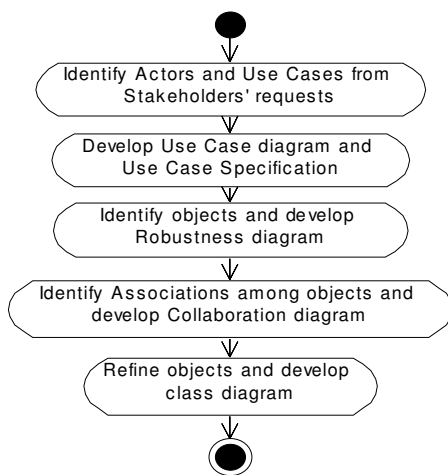


Fig. 1 The workflow of UCDA tool

3. UCDA: Use Case driven Development Assistant

3.1. Overview

UCDA (Use Case driven Development Assistant) is a tool that helps developers to develop use case models, robustness diagrams, collaboration diagrams and class diagrams; and to visualize these models using the Rational Rose tool. UCDA supports the UML standards. A freely available natural language parser is integrated with UCDA tool to parse stakeholder requests. The various guidelines to extract the actors, use cases, objects and classes are applied to the parser output.

UCDA has the following features. It:

- Uses a natural language parser to parse stakeholder requests. Basically the parser analyzes the sentence and tags each word with its part-of-speech.

- Recognises certain complex sentences and simplifies them.
- Identifies actors and use cases and develops use case diagrams.
- Generates XML files containing the details of the models.
- Generates the models in Rational Rose.
- Assists the user with filling in the use case specification template.
- Validates the use case specification document.
- Reads the use case specification to identify objects and their associations; and develops robustness diagrams.
- Identifies messages passed between objects and develops collaboration diagrams.
- Generates and validates the class model.

3.2. Architecture

The architecture of UCDA is shown in Fig. 2.

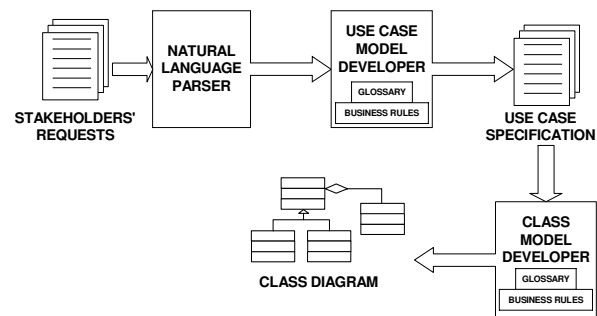


Fig. 2 Architecture of UCDA

UCDA consists of two main components: The Use Case model developer and the Class model developer.

The use case model developer helps the stakeholder to specify the requirements of the proposed system and to identify actors and use cases. These actors and use cases can be exported to the modeling tool. The modeling tool used in the UCDA tool is Rational Rose. The changes made to the model in Rational Rose affects the elements in UCDA tool.

Use cases can be realized by the Class model developer. Objects and messages between them are identified from the use case specifications. The tool can generate robustness diagrams and collaboration diagrams in Rational Rose. The behaviour described in use case specifications can be distributed to the analysis classes. The analysis class model is the final output of the tool.

3.3. Use Case Model Developer

The various features of use case model developer component are:

- Read text entered by stakeholder and assign part-of-speech to words in the text.
- Study the structure of the sentence and check if it is a complex sentence (e.g., sentences with more than one verbs or sentences with conjunctions)
- If the sentence is complex, reduce the complexity by splitting it into simpler sentences.
- Retrieve the subject and predicate from the simple sentences.
- Identify the modifiers (e.g., adjectives, auxiliary verbs) associated with the subject and predicate.
- Filter the modifiers and check if subject exists in the glossary. If subject exists then subject is the actor and predicate is the use case.

Kurt Bittner and Ian Spence provide a questionnaire to identify actors and use cases [5]. In UCDA, the structure of the sentence is considered for automating actor and use case identification.

The following rules are applied for automating actor and use case identification.

1. If the Subject of a sentence is a noun or noun phrase consisting only of nouns, and this noun/noun phrase is found in the glossary then the noun/noun phrase is an actor.
2. If system is the Subject of the sentence, then it is not a valid actor.
3. If the Subject of a sentence is an actor, then the following predicate (P) forms are valid use cases.
P: V; P: V/NP; P: V/PP
4. If the Subject of a sentence is an actor and the predicate is of the form: P: VP/NP1/"from"/NP2 then VP/NP1 is the use case and NP2 is an actor and an association exists from the actor to the use case.
5. If the Subject of a sentence is an actor and the predicate is of the form: P: VP/NP1/"to"/NP2 then VP/NP1 is the use case and NP2 is an actor and an association exists from an actor to the use case.

V: Verb; NP: Noun Phrase; PP: Prepositional Phrase

The next task of the use case model developer component is to generate the use case specification document. Each use case is associated with a use case specification document. UCDA provides the interface for the user to enter use case specification details according to the use case specification template (shown in Table 1).

Table 1. Use Case Specification template

1.	Use Case Name
1.1	Brief description
2.	Flow of Events
2.1	Basic Flow
2.2	Alternative Flow
3.	Special Requirements
4.	Preconditions
5.	Post conditions
6.	Extension Points

This document is parsed by the class model developer component to identify objects. So the statement structures should be simple (shown in Fig. 3).

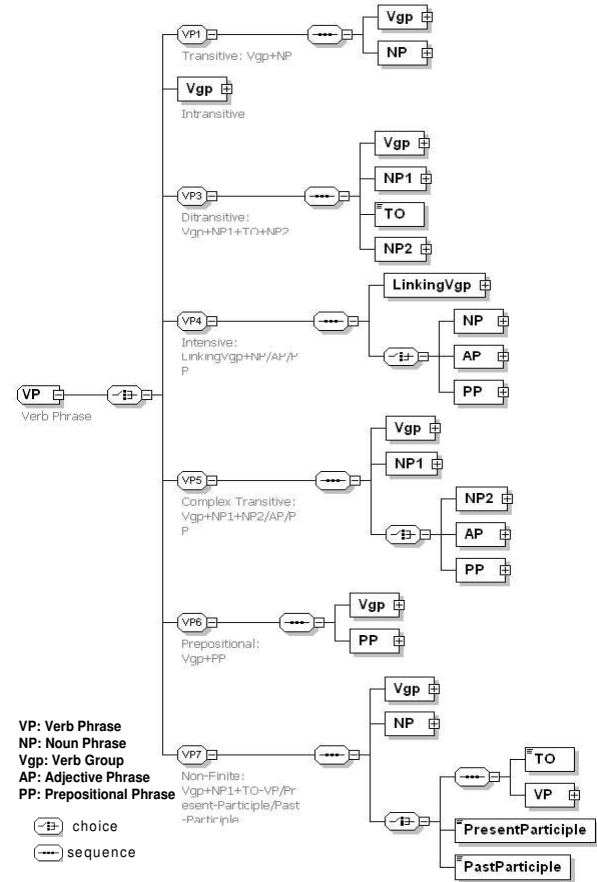


Fig. 3 Possible statement structures

UCDA verifies that the statements entered by the user conform to the above statement structures. Conditional statements start with an IF statement, iterative statements with a WHILE statement and concurrency statements with a CON statement. Pronouns are replaced by concrete nouns and Passive voice is reconstructed to be active. The tool verifies the document based on completeness, complexity and structure.

3.4. Class Model Developer

To develop the class model from the use case specifications, we summarized the relationships between syntactic structures of natural language and semantic associations of objects in the models. Fig. 4 shows a model of the actions in actor-system interaction [6]. Four types of behaviour are included in the model. The relationships between the behaviour types and the associations of stereotype objects are listed in Table 2.

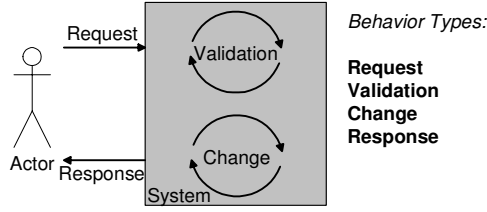


Fig. 4 An actor-system interaction model

Table 2. Relationships between behaviour types and associations between stereotype objects

Behaviour Type	Association
Request	
Validation	and
Change	
Response	

: actor, : boundary object, : control object, : entity object

We identified the relationships between all statement structures and behaviour types, and represented them in 17 rules for object and message identification [7]. Because of the length of this paper, we only demonstrate one rule for transitive structure shown in Fig. 5, where NP represents a noun phrase; VPss represents a verb phrase with the statement structure; PP represents a prepositional phrase; Vgp represents a verb group; and Prep represents a preposition.

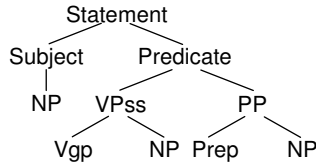


Fig. 5 The structure of a transitive statement

The rule for object identification is:

Rule: If the structure of a statement is transitive (as shown in Fig. 3), and Subject/NP/Noun(head) is an actor, then this statement corresponds to the Request behaviour type. Predicate/PP/NP/Noun(head) is a boundary object if it exists in the glossary, and Predicate/VPss/NP/Noun(head) is an entity object if it exists in the glossary. If there are two objects or an actor and an object in one statement, an association between them is identified. To generate the collaboration diagram, the messages contained in one scenario are identified. The corresponding rule for message identification is:
Rule: If Subject/NP/Noun(head) is an actor, and Predicate/PP/NP/Noun(head) is a boundary object, then the action is Predication/VPss/Vgp/NP/Noun(head) + Predication/VPss/Vgp/NP/Noun(head), the sender is Subject/NP/Noun(head) and the receiver is Predicate/PP/NP/Noun(head).

The responsibilities of the classes can be identified from the messages in the collaboration diagrams. Each message consists of a sender, a receiver, and an action. The receiver has the responsibility for the execution of the action. The messages in collaboration diagrams are transformed to the classes' responsibilities in this way.

Composition, generalization and aggregation relationships are to be identified in the class model of the system under development.

Rule: If one use case includes another use case, then a composition relationship is likely to exist between the core control classes identified from the use cases.

Rule: If one use case has a generalization relationship with another use case, then a generalization relationship is likely to also exist between the core control classes identified from the use cases.

We propose a method to validate the analysis model, especially the robustness diagrams. There are some constraints for objects and associations in a robustness diagram according to its semantics. The rules listed in Table 3 are derived from the constraints and used for robustness diagram validation.

Table 3. Rules for robustness diagram validation

Case	Validation	Suggestion
	Not allowed.	
	Allowed	
	Not allowed.	
	Not allowed.	
	Not allowed.	
	Allowed.	
	Not allowed.	
	Allowed.	
	Allowed.	
	Not allowed.	

: actor, : boundary object, : control object, : entity object

4. UCDA: Case Study

This section uses the ATM system specification to show the working of the proposed system. The functional description of an ATM system is:

"The customer inserts the cash card in the machine. Customer can withdraw cash from an account. The bank approves the transaction. In addition, customer can deposit the amount. Customer can transfer amount between accounts. Customers can check the balance in

the account. The customer can cancel the transaction at any time.”

The natural language parser parses the specification. Based on the rules given in Section 3.3, the system identifies the actors and use cases, and generates an XML file containing actors and use cases. The tool reads the XML file and generates the corresponding diagrams in the Rational Rose tool. The use case diagram generated by the UCDA tool is shown in Fig. 6. The association between actors and use cases are shown in use case diagram.

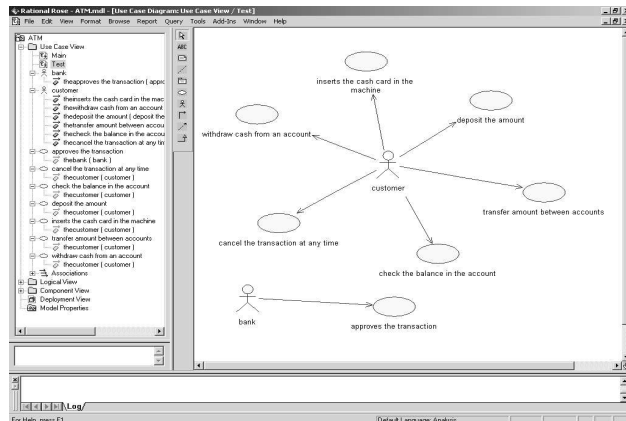


Fig. 6 Use Case diagram generated by UCDA

We compared the use case diagram generated by the tool with that of software engineering graduate students having considerable knowledge in this area. The tool is evaluated based on the time taken to develop the use case diagram and the number of correct actors and use cases identified. The tool generates the use case diagram much faster compared to students and the number of correct actors and use cases generated by the tool are also higher.

We also compared the use case diagram generated by the tool with the use case diagram provided by experts. We found that the tool has identified 100% of the actors and 70% of the use cases.

To complete the use case model, each use case is associated with a use case specification document. Fig. 7 shows the user interface for writing use case specification details. The tool provides the template to document conditional, iterative and concurrent statements. The tool verifies the sentences entered by the user according to the format specified in Fig. 3. The tool generates the document in word format and XML format. The use case specification document for the “withdraw cash” use case is specified in Fig. 8. The CREWS project suggests several guidelines for writing the use case specification document [9]. Our tool implements these guidelines and templates.

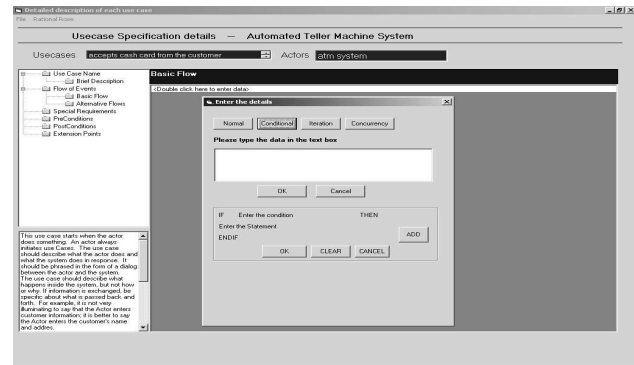


Fig. 7 Use Case Specification Interface Screen

Actors: customer, bank

Flow of Events:

Basic Flow:

1. the system start withdrawal transaction;
2. the customer select the account on the customer console;
3. the system get the account from the customer console;
4. the customer select the amount on the customer console;
5. the system get the amount from the customer console;
6. the system generate the withdrawal transaction information;
7. the system send the withdrawal transaction information to the bank;
8. the bank send the withdrawal transaction approval to the system;
9. the system dispense the cash in the cash dispenser;
10. the customer get the cash from the cash dispenser;
11. the system record the withdrawal transaction information into the log;
12. the withdrawal transaction end;

Alternative Flow:

If the bank do not approve the withdrawal transaction, then

1. the system display an error message on the customer console;
2. the system record the withdrawal transaction information into the log;
3. the withdrawal transaction end;

Fig. 8 Use Case Specification of ‘withdraw cash’

The specification is processed using the methodology discussed in Section 3.4. Fig. 9 shows the environment for use case realization including Rational Rose.

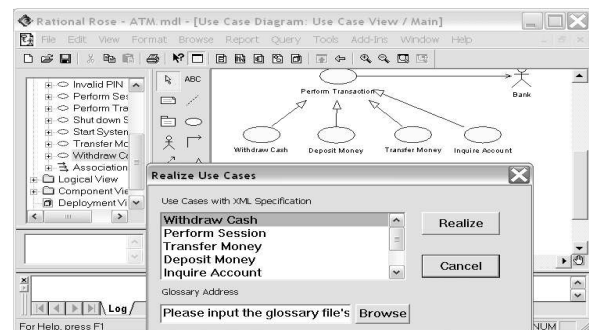


Fig. 9 The environment for use case realization

Fig. 10 shows the robustness diagram that is generated from the use case specification. The validation of the robustness diagram shows that there should be a boundary object between the bank and the Withdrawal transaction class. The use case specification is reviewed and steps 7 and 8 in the basic flow are revised as follows:

7. the system send the withdrawal transaction information to the network connection;

8. the bank get the withdrawal transaction information from the network connection;
9. the bank send the withdrawal transaction approval to the network connection;
10. the system get the withdrawal transaction approval from the network connection;

A new robustness diagram is generated according to the revised use case specification and shown in Fig. 11.

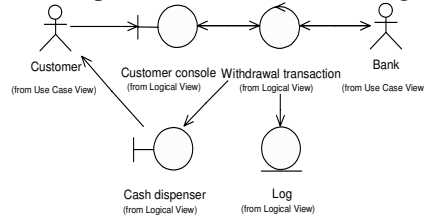


Fig. 10 Robustness diagram

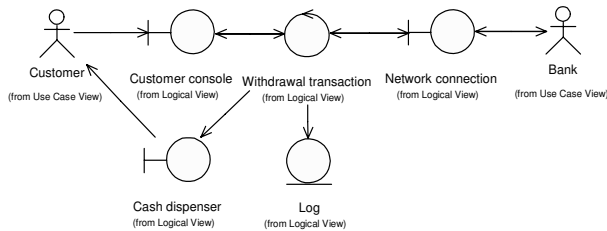


Fig. 11 New Robustness diagram

The collaboration diagram is shown in Fig. 12, and the class diagram containing the identified classes from the use case is shown in Fig. 13. All the diagrams are automatically generated by UCDA.

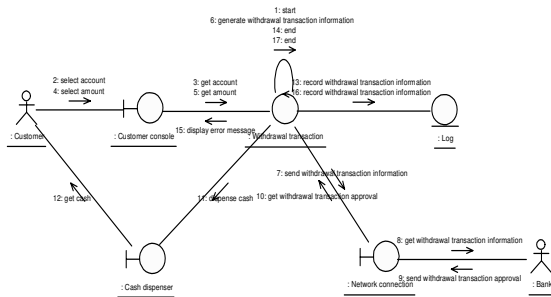


Fig. 12 Collaboration diagram

5. Conclusion

Identifying objects and classes is a challenging task in software engineering. In this paper, we presented a methodology to develop a class model from natural language requirements and its implementation in the UCDA tool. The methodology focuses on generating intermediate artifacts such as use case diagrams, robustness diagrams and collaboration diagrams.

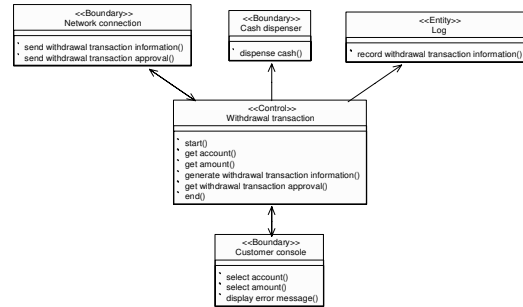


Fig. 13 Class diagram

We compared the performance of the tool with the software engineering student's performance provided the student already knows about OOAD. We found that the tool generates the better class model in less time compared to the students. By automating these tasks, software design effort and cost can be reduced. Furthermore, this tool provides valuable guidance to novice software designers.

References

- [1] L.C. Niba, "The NIBA Workflow: From textual requirements specification to UML-schemata", Proceedings of International Conference on Software & Systems Engineering and their Applications, ICSSEA 2002, Paris, December 2002.
- [2] Scott P. Overmyer, Benoit Lavoie, Owen Rambow, "Conceptual Modeling through Linguistic Analysis Using LIDA", Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 401-410, May 2001, Toronto, Ontario, Canada.
- [3] Rational Unified Process
<http://www-306.ibm.com/software/awdtools/rup/>
- [4] Romi S. Wahono, Behrouz H. Far, Jingde Cheng, "A Framework of Object Identification and Refinement Process in Object-Oriented Analysis and Design", Proceedings of the 1st IEEE International Conference on Cognitive Informatics, ICCI 2002, Calgary, Canada, 2002.
- [5] Kurt Bittner, Ian Spence, "Use Case Modeling", Addison Wesley, 2002.
- [6] Alistair Cockburn, "Writing Effective Use Cases", Addison-Wesley, 2000.
- [7] D. Liu, "Automating Transition from Use Cases to Class Model", Master Thesis, University of Calgary, Calgary, 2003.
- [8] Grady Booch, "Object-Oriented Analysis and Design with Applications", Addison-Wesley, 1994.
- [9] Karl Cox, Keith Phalp: "Use Case Authoring: Replicating the CREWS Guidelines Experiment", Int. Journal of Empirical Software Engineering, Issue 5, 245-267, 2000.

Using A Scenario Specification Language to Add Context to Design Patterns

Reginald L. Hobbs
Army Research Laboratory
hobbs@arl.army.mil

Abstract. Software designers have used design patterns and frameworks to describe reusable architectures for development. The approach is to capture the best practices in software architecture to create a toolkit of solutions for the practitioner. Incorporating additional semantics within the problem/solution space can further enhance these pattern descriptions. This context can be inserted using narrative. This paper describes a research study that defined a scenario specification language based on a narrative conceptual model. This language can be merged with representations of design patterns, which describe the static, dynamic, and collaborative attributes of software solutions. This will enable pattern descriptions to be narrative exemplars that could be used directly in design tasks.

1. Introduction

A design pattern is a description of communicating objects and classes that are customized to solve a general design problem in a particular context.[5] A pattern has four essential elements: 1) a name, 2) the description of the problem, 3) a solution and 4) the costs and benefits of applying the pattern.

Software engineers are interested in using design patterns to create a repository of solutions for software development. The concept is similar to the sourcebooks used by other engineering disciplines, such as civil or electrical engineering. When faced with a particular design problem, civil engineers can consult from a catalogue of possible techniques. These solutions are based on known successful methods from the industry. The problem with this approach for the software engineering discipline is that software engineering involves analyzing abstractions that are not based on physical systems. Software design patterns could be normative at best, describing suggested solutions given enough similarities in the problem space. However, there are several motivating factors that make it worthwhile to identify these patterns: 1) Success is more important than novelty, 2) Clarity of communication should be emphasized, 3) Qualitatively validating of concrete

solutions, 4) Good patterns arise from practical experience, and 5) the incorporation of human factors into software development. [11]

The benefits of design patterns are constrained by the difficulty in applying them in real-world situations. Many design patterns are unnecessarily difficult for the average designer to learn. There is also the problem of making the pattern classifications useful to the practitioner. Some of the categorizations do not appear to map to the mental models used by the average developer. [3] Using a scenario-based technique, particular one based on narrative, would enable the patterns to better fit into the terminology and knowledge base of the developer.

The next section is a brief discussion on design patterns. Section 3 is a description of the narrative ontology used to develop a scenario specific language. Section 4 describes approaches to merging scenarios with design patterns. Finally, there is an example of creating a story pattern using a XML pattern language with the scenario grammar.

2. Overview of Design Patterns

An architect, Christopher Alexander, initially developed the concept of a pattern language. Alexander wished to capture the recurring aesthetic features of a living space as well as detail standard design solutions. These architectural patterns were created as an aid to the participatory design process, involving user requirements as well as technical guidelines. [4]

Software design patterns and frameworks were described to support reusable architecture and detailed design, respectively. A framework is a set of components that provide a reusable infrastructure for a family of related applications. Pattern descriptions are often independent of implementation details, whereas, frameworks are semi-complete applications that provide domain-specific functionality. Patterns are abstract representations of the problem space that could utilize a particular instance of a framework for portions of the solution. By the same

token, a framework may contain several different patterns within its implementation. For the purpose of this discussion, we will focus on incorporating scenario information within patterns. This will allow the flexibility inherent in scenario analysis to take advantage of the abstract nature of software patterns. However, scenarios could be associated with frameworks to describe alternate configurations of components within a domain.

There are 3 major categories of patterns: creational, structural, and behavioural. *Creational* patterns are those that deal with initializing and configuring classes and objects. *Structural* patterns seek to separate interface from implementation issues in design. Finally, *behavioural* patterns deal with interactions and collaborations among collections of classes and objects. Within each category, there are numerous identified patterns derived from recurring design tasks. For example, the behavioural patterns are:

- [1] Chain of Responsibility - object requests are routed to the responsible service provider
- [2] Command - requests are treated as objects
- [3] Interpreter – language interpreter for a grammar
- [4] Iterator – An object accesses aggregate elements sequentially
- [5] Mediator – Object coordinates interactions between associate objects
- [6] Memento – System snapshot captures and restores object states
- [7] Observer – Dependent objects update when a subject changes state
- [8] State – Object behaviour depends upon its current state
- [9] Strategy – Abstraction for allowing the selection from different algorithms
- [10] Template Method – Algorithm with steps supplied by a derived class
- [11] Visitor – Operations are applied to a heterogeneous object

3. A Narrative Approach to Scenario Specification

Many practical disciplines make use of scenarios to help practitioners make effective decisions. System designers and policy makers use scenarios to assist them in making design and policy decisions. Scenarios are used throughout software development to examine alternate design decisions and requirements.[2] Requirements engineering, the process of determining requirements for a proposed system has used reasoning techniques about scenarios to help generate and evaluate specifications.[1]

Scenario-based methods seek to reduce the complexity of design by focusing on the structure and the dynamics of the problem domain.[1] These "what-if" studies allow software developers to refine the design and requirements of a system before (and sometimes during) implementation. By catching potential system errors and design problems early in system evolution, costly redesigns are minimized.

Not only are scenarios of practical importance, they appear to play a fundamental role in comprehension. Cognitive scientists note that narrative is central to processes of explanation, inference, and interpretation.[10] Scenarios codify and externalize algorithmic mental models and thought experiments.[6] Decision-makers may test hypotheses by constructing "what-if" scenarios on top of a baseline description of the situation under consideration. Whether these scenarios are described, enacted through role-playing, or simulated computationally, whether they faithfully represent a real phenomenon or merely provide the outline sketch of a possible course of action, the intention is the same: to clarify the relationships among actions and features of a situation and to understand better the consequences of actions.

3.1. Narrative Ontology for Scenarios

The difficulty in developing a meta-model for scenarios rests in the numerous styles, categories, and implementations of scenarios. This difficulty can be handled by describing an abstraction of scenarios separate from any specific context. This abstraction came about by describing all scenarios as stories (narrative). They are stories about what was done, what is being done, or what can be done. These different forms of narrative can be used for problem solving, training, entertainment, and any other activity that involves decision-making. Studying different forms of narrative, from film editing, screenwriting, use cases, literature, and cognitive science, would isolate the essential elements of scenarios with respect to problem solving. This narrative morphology can then be used to establish a conceptual model, rules, and transformations for scenarios. [7]

Figure 1 shows the narrative model represented as a UML (Unified Modeling Language) class diagram. The diagram establishes a hierarchy of story elements and the associations among them. Each element is a first-order object that can be manipulated within scenarios.

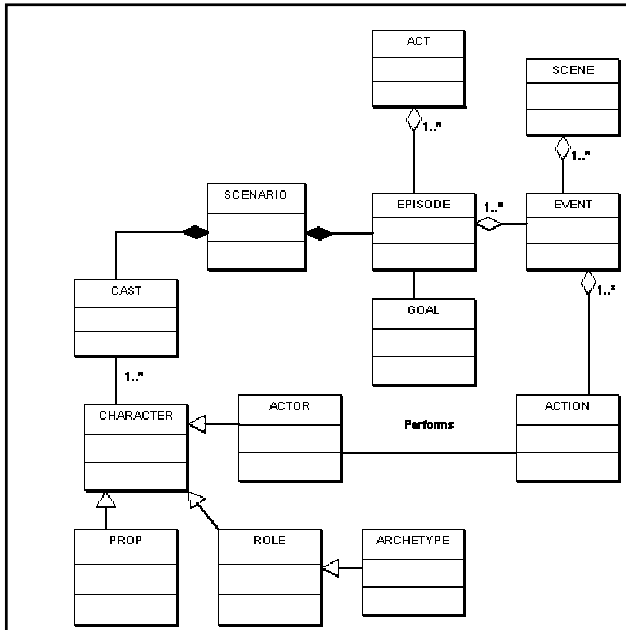


Figure 1 Scenario Conceptual Model

The fundamental building block of scenario is the *action*. An action is anything that happens in the context of the story. It could consist of characters communicating, the physical movement of an object, or a change of state. Although actions are performed by actors and cause changes to occur in objects, it is the actions themselves that make a scenario a narrative rather than a mere description of a situation.

Meaningful sequences of actions make up *events*, and predetermined lists of events are combined into *episodes*. Episodes are goal-based objects, with the individual events occurring to support sub-goals. Each scenario can be described as a succession of episodes (goals) that were either achieved or thwarted. It is this goal-based, hierarchical nature of the scenario structure that affords manipulation of the story.

3.2. Scenario Mark-up Language (SCML)

The goal is to represent the narrative model in a form that is general enough to describe the structure of any scenario, yet expressive enough to support different external views of the data. The choice of a mark-up language supports both these objectives. Mark-up languages describe a document's structure, leaving presentation details to the capabilities of the structure-aware applications.

To make it convenient to create different representations of the same scenario content, the concept of a hypermedia scenario document, or *hyperscenario*, is introduced. [8] Hyperscenario documents are scenarios created using a

mark-up language approach. The semantic information within narrative and the relationship/hierarchy between story elements is represented structurally with tags and attributes in a scenario specification grammar. The XML implementation of the grammar is called *SCML* (Scenario Mark-up Language). [7] The dynamic movement through narrative that is necessary for comprehension and decision-making is defined using embedded hyperlinks, representing alternate story paths. Hyperlinks among text, graphics, and other multimedia elements support multiple perspectives and methods for discussing scenarios. The structural narrative model can be created in SCML, while the rendering and navigational aspects can be handled by the appropriate scenario application.

Specific mark-up languages are defined from XML by creating a *DTD* (Document Type Definition).[12] The DTD defines allowable components and structures for documents of its type. Designing SCML therefore involved mapping the narrative ontology to an XML schema. Major story components, such as episodes, goals, and actions are all reflected as elements in the SCML DTD. Another important aspect of SCML is the support for the link strategy. Within an XML-based language, the developer can define link types using XLL (XML Linking Language). SCML links are bi-directional. Extended links can target a selection of possible document artifacts, as opposed to a single file. External programs can be executed by activating a link using this method.

4. Defining Story Patterns

Figure 2 depicts two approaches for incorporating narrative information with design patterns. In the Design-Patterns-with-Scenarios approach, each of the smaller items, S_i , represent scenario *variants*. A scenario variant is an alternate form of the same scenario. For example, let's assume we are describing a scenario calling "Getting To The Airport". One instance of this scenario could involve driving through the city, taking the appropriate exit for the airport. A variant on this scenario would be hiring a taxi or shuttle as transportation directly to the airport. With respect to software, each variant within a pattern could represent scenario descriptions of alternate implementations of the pattern. These variants could be due to programming language, networking, or even platform differences. The second approach is the Scenario-with-Design-Patterns method. In this method, the P_j represent several possible solutions for a particular design problem. For example, a scenario could be constructed to represent the requirements of a software system's user interface, based on a particular activity to be accomplished.

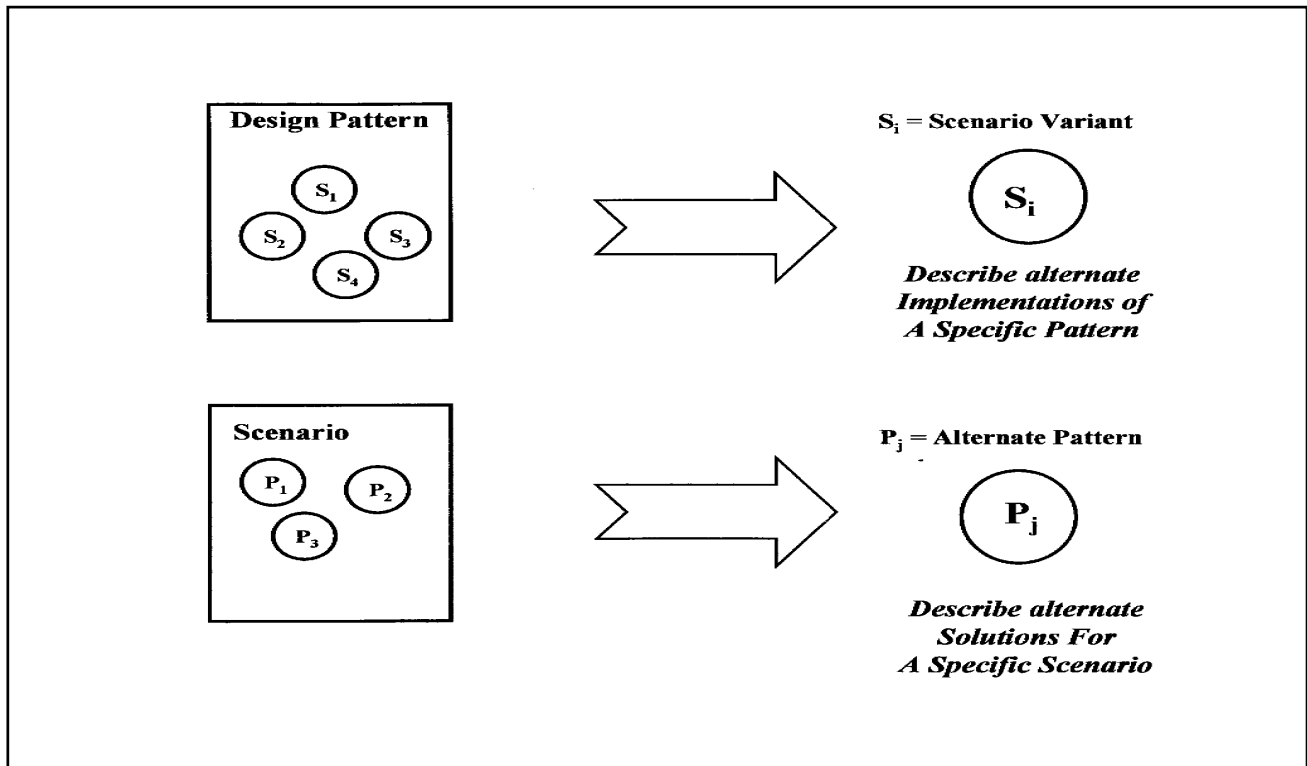


Figure 2 Merging Design Patterns with Scenarios

5. Incorporating Scenario Context within Pattern languages

Since the beginnings of discussion on the utility of design patterns, there have been efforts to create languages for representing pattern artefacts. They were designed to allow for the direct manipulation of the pattern descriptions in design tools, particularly during requirements analysis. There has also been research into the design of usability pattern languages that could assist the user-centered design process. [9] Most recently, there are attempts at defining XML-based pattern languages that would be useful across the entire software life cycle. These languages could also leverage the growing number of XML technologies and environments.

One example pattern language is *DPML* (Design Pattern Mark-up Language).[13] This language was developed as part of a research effort to automatically detect design patterns from source code. The researchers viewed design patterns as higher-level abstractions of the object-oriented design within the code. Recognizing these patterns would serve as an aid to program comprehension, code documentation, and validation. DPML was used to create a pattern library, consisting of most of the standard

software design patterns as outlined in the descriptions by Gamma et.al.[5] C++ source code was analyzed to create a class diagram, call graph, and object creation graph. These were then matched against DMPL patterns formatted as an XML DOM (Document Object Model) tree structure. The algorithm was used on four open-source C++ projects. Of 26 patterns searched for in over 3 million lines-of-code, 15 patterns were detected, with 978 different instances occurring.

Figure 3 is an example of how the narrative structure available in SCML could be used with DPML. In this particular situation, we are taking the Scenario-with-Patterns approach. The scenario may contain several patterns that represent alternate solutions within the story. The scenario is a description of the process of determining networking requirements for a proposed system. The purpose of the scenario is to examine the risks/benefits of different network implementations. Here, each design pattern is considered to be a character within the scenario. An XML Namespace is used to incorporate DPML elements within the hyperscenario structure. *Namespaces* are a technique to import information from external languages without conflicting with the grammar of the current language.[12] The 'DPML:' prefix identifies those elements that are not part of the SCML grammar. The classes contained in the pattern are described in the element definition. The

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE hyperscenario SYSTEM "scml.dtd">
<hyperscenario xmlns:DPML="dpml.dtd"
  title="Network Requirement Analysis"
  purpose="Examine Alternatives for Implementing System Network"
  logline="Risk Assessment of Network Strategies">
<cast>
<character><actor><DPML:DesignPattern='Proxy'>
  <DPML:Class id='id10' name='Subject' isAbstract='true'>
    <DPML:Operation id='id11' name='Request'
      isVirtual='true'><DPML:hasTypeRep ref='id50'>
    </DPML:Operation></DPML:Class>

    <DPML:Class id='id20' name='Proxy'><DPML:Base ref='id10'>
    <DPML:Aggregation ref='id30'> <DPML:Operation id='id21'
      name='Request' isVirtual='true'>
    <DPML:defines ref='id11'> <DPML:calls ref='id31'>
    <DPML:hasTypeRep ref='id50'></DPML:Operation>
    <DPML:Attribute id='id22' name='realSubject'>
    <DPML:hasTypeRep ref='id52'>
    </DPML:Attribute></DPML:Class>

    <DPML:Class id='id30' name='RealSubject'>
    <DPML:Base ref='id10'>
    <DPML:Operation id='id31' name='Request' isVirtual='true'>
    <DPML:defines ref='id11'><DPML:hasTypeRep ref='id50'>
    </DPML:Operation></DPML:Class>
    </DPML:DesignPattern></actor></character>
  .
  .
  .
</cast>
<episode id="000.01" name="Network Service Request">
<goal>Handle External Network Requests</goal>
<scene><setting>Local Area Network</setting>
<event name="Establish Network Connection">
  .
  .
  .

```

Figure 3 Design Pattern Mark-up Language (DPML) and SCML

example code for the Proxy pattern is a modified version from the DPML study. [13] There would be a character entry for each potential pattern that could be used as a network solution.

Each episode of the scenario analysis associates networking alternatives against a stated goal of the network infrastructure. In this example, the episode would examine how service requests would be handle on the local network by each of the characters (patterns). Within each episode, there are a series of events that occur and should be interpreted and handled. For example, the event of establishing an initial network connection. There are domain-specific actions associated with the patterns in the scenario, some of which could be derived from the operations defined within the pattern. The completed scenario captures design decisions and

establish a way of associating non-functional requirements with the system.

6. Conclusion

Scenario-based design and software design patterns are approaches to handle the complexity and uncertainty inherent in software development. Scenario-based methods are used to capture and track design decisions made during requirements analysis. Design patterns attempt to leverage the best practices in software engineering by recognizing recurring problems and their solutions. This paper discussed a method that could be utilized to merge both techniques, creating reusable story patterns that assign further context to proposed design solutions.

References

- [1] J. S. Anderson and B. Durney, "Using Scenarios in Deficiency-driven Requirements Engineering". In Proceedings of the IEEE International Symposium on Requirements Engineering, San Diego, CA, 4-6 January 1993.
- [2] J. Carroll. "Making Use: Scenario-based Design of Human-Computer Interactions", MIT Press. Cambridge, MA. 2000.
- [3] M. Cline, "The Pros and Cons of Adopting and Applying Design Patterns in the Real World", Communications of the ACM, Vol. 39, No. 10, 1996, pp. 37-39.
- [4] A. Dearden, J. Finlay, E. Allgar, and B. McManus, "Using Pattern Languages in Participatory Design", in Proceedings of the Participatory Design Conference 2002, Malmo, Sweden, 23-25 June 2002.
- [5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Indianapolis, IN. 1995.
- [6] D. Gooding, "The Procedural Turn; or, Why Do Thought Experiments Work", Cognitive Models of Science. University of Minnesota, Minneapolis, 1992. Pages 45-76.
- [7] R. Hobbs, "Sharing Stories: Using Narrative for Simulations Interoperability", in Proceedings of the 2003 Spring Simulations Interoperability Workshop, Orlando, FL., 30 March - 4 April 2003.
- [8] R. Hobbs, "Hyperscenarios: A Framework for Active Narrative" in Proceedings of the 38th Annual ACM Southeast Conference, Clemson, SC., 7-8 April 2000.

- [9] M. J. Mahemoff and L. J. Johnston, "Pattern Languages for Usability: An Investigation into Alternative Approaches", in Proceedings of the Asia-Pacific Conference on HCI 1998 (APCHI), Los Alamitos, CA, 15-17 July 1998.
- [10] N. Nersessian, "How Do Scientists Think?", in Cognitive Models of Science. Univ. of Minnesota, Minneapolis, 1992. pages 3-44.
- [11] D. Schmidt, R. Johnson, and M. Fayad, "Software Patterns", Communications of the ACM, Vol. 39, No. 10, 1996, pp. 37-39.
- [12] World Wide Web (W3C) Consortium. "Extensible Markup Language (XML) 1.0." W3C Recommendation. See <http://www.w3.org/TR/1998/REC-xml-19980210>.
- [13] B. Zsolt and R. Ferenc, "Mining Design Patterns from C++ Source Code", In Proceedings of the International Conference on Software Maintenance (ICSM '03), Amsterdam, The Netherlands, 22-26 September 2003.

Visualizing the evolution of software using softChange

Daniel M. German, Abram Hindle and Norman Jordan
Software Engineering Group
Department of Computer Science
University of Victoria
{dmgerman,abez,njordan}@uvic.ca

Abstract

*A typical software development team leaves behind a large amount of information. This information takes different forms, such as mail messages, software releases, version control logs, defect reports, etc. **softChange** is a tool that retrieves this information, analysis and enhances it by finding new relationships amongst it, and allows users to navigate and visualize this information. The main objective of **softChange** is to help programmers, their management and software evolution researchers in understanding how a software product has evolved since its conception.*

Keywords Software evolution, software trails, CVS, visualization, softChange.

1. Introduction

Many software projects use a version control repository to record the the evolution of their source code. These repositories keep track of every change to any source file of the project, including metadata about the change, such as author and date when it happened. Over time, the amount of revisions to a project become enormous. For example, the Mozilla project is composed of 35,000 files which have been modified 450,000 times in 5.5 years of development (from March 1998 to Aug. 2003) by 500 different developers.

CVS, the Concurrent Versioning System, is arguably the most widely used version control management system available in the market and has become a de-facto standard in the development of open source projects.

While CVS is a very powerful tool, it provides many barriers to the extraction and visualization of valuable information. CVS commands are cryptic and their output formats are not easy to understand. CVS queries often produce an excess of information which is hard for the frustrated de-

veloper to sift through. General summaries are rarely provided. Furthermore, CVS does not provide an alternative to browsing through its information.

CVS is built around a group of command-line programs. Several GUI applications have been built around (winCVS, tkCVS, cvsWeb, LinCVS, Pharmacy, gCVS, etc) and some integrated development environments (such as Eclipse) provide a GUI to CVS. In all these cases, the tools are created around the CVS commands and options, providing nothing more than a fancy GUI to the actual commands.

One of the main disadvantages of CVS is that it is not transaction oriented. In other words, when a developer proceeds to “commit” a group of changes to a number of files, CVS does not keep track of all the files modified by this commit operation. It treats each change to a file independently of the other files included in the commit. After the commit has taken place, CVS does not know which files were modified together. This information, however, is important because it highlights coupling amongst files: if two files are modified at the same time, it is because they share *something* in common. We refer in this paper to an commit operation as a *modification request* (MR). A MR is therefore a collection of revisions to files that are modified at the same time.

The information stored in the CVS repository is quite valuable as it can help answering many questions. For instance, it can assist developers in knowing who has modified which files and when; it can also help the administration in trying to understand the modification patterns of the project and the way the different team members interact. Finally, it can help in the recovery of the evolution of the project (as we reported in [6]). For example, developers can ask the following questions [11]:

- What happened since I last worked on this project?
- Who made this happen?
- When did the change take place?
- Where did the change happen?

- Why were these changes made?
- How has the file changed?
- What methods or functions were changed?
- What is the frequency of change?
- What files changed?
- Who is working on each modules?

Administrators, on the other hand, are interested in higher level questions and metrics such as:

- How often does a programmer complete a MR?
- How much does the programmer change per MR
- What kind of commits does one programmer do?
- How much changed between each release?
- How many bugs are fixed and found after a stable release?
- What kind of modifications are done at a certain time?
- When was a module stabilized?
- What is the daily LOC count for each programmer?
- When is a module actively being developed and maintained?

We define *software trails* as information left behind by the contributors to the development process, such as mailing lists, Web sites, version control logs, software releases, documentation, and the source code [6]. In this paper we describe **softChange**, a tool that mines software trails from CVS repositories, then enhances this data with some heuristics in order to recover higher level information, such as rebuilding MRs. Each MR is analyzed in order to know what type of changes took place; such as adding new functions, reorganizing source code, adding comments to the code only, etc. After extraction and analysis, **softChange** provides a graphical and hypertext representation of this information. This paper is divided as follows: previous work is described in section 2; section 3 describes the architecture of **softChange**; section 4 describes the visualization features of **softChange**; we end describing our experiences using **softChange**, our conclusions, and future work.

2. Previous Work

The two most commonly used hypertext front ends to CVS are Bonsai [8] and lrx [7]. They provide a Web interface to the CVS repository and isolate the user from the complexities of the CVS commands (the man page of CVS is 9000 words long, approximately 3 times the length of this paper). Both tools allow the user to inspect the history of any given file in the project and neither of them attempts to enhance the software trails available in the repository.

Xia is a plugin for Eclipse for the visualization of CVS repositories[11]. Xia recovers relations available in the logs of a CVS repository and allows the user to navigate them. It uses squares to represent files, their revisions and developers, and lines to represent the relationships between them. Xia has two main limitations. The first is that Xia relies on the Eclipse API to access the CVS repository. Every time Xia wants to create a view, it queries the CVS repository in order to retrieve the necessary data. This becomes a very expensive operation making Xia extremely slow in large CVS repositories. The second limitation is that Xia operates at the revision level, not at the MR level.

Hipikat aggregates many sources of information such as bugzilla, the CVS repository, mailing lists, emails etc and provides a searchable query interface[1]. The purpose of Hipikat is to "recommend software artifacts" rather than summarize and visualize them. Thus Hipikat is much like Google for a software project. One interesting feature of Hipikat is that it correlates software trails from different sources, inferring relationships between them.

Liu and Stroulia have developed **JReflex**, a plug-in for Eclipse for instructors of software engineering courses. **JReflex** helps the instructor to monitor how different teams of students developed a term project by using their CVS historical information [9]. It is designed to compare the differences in development styles in different teams, who does what, who works on what part of the project, etc. **JReflex** is intended to be a management oriented tool for browsing the CVS historical data. **JReflex** does not enhance the information available in CVS.

Fisher and Gall have described a CVS fact extractor in [2]. In it they describe the main challenges of creating a database of CVS historical data and then use it to visualize the interrelationships between files in a project [3].

3. softChange Architecture

softChange is composed of four main components, depicted in figure 1.

- **Software trails repository:** At the core of **softChange** lies a relational database that is used to store all the historical information.

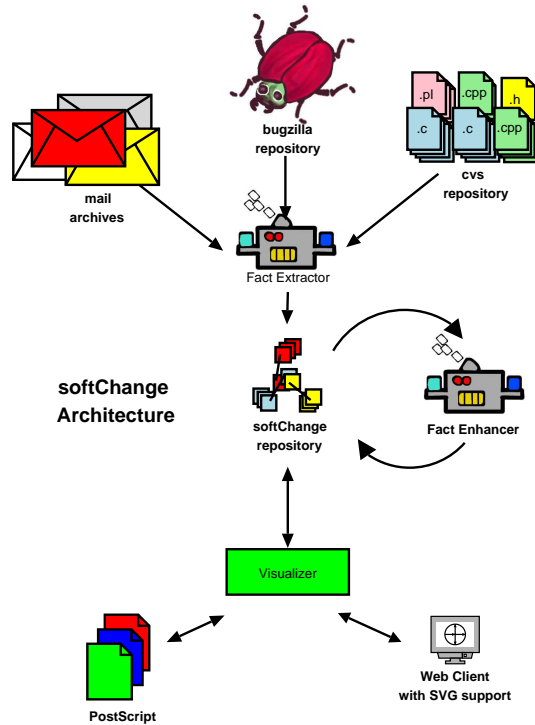


Figure 1. Architecture of AUSS

- **Software trails extractor:** In a typical software development project, software trails originate from many different sources: CVS historical data, email messages, bug reports, ChangeLogs, etc. The purpose of **softChange** trails extractor is to retrieve as many software trails as possible. Currently, **softChange** is able to retrieve trails from CVS, from ChangeLogs, from the releases of the software (the tar files distributed by the software team) and from Bugzilla.
- **Software trails analyzer:** Once **softChange** has extracted the software trails, it proceeds to use this information to generate new facts. For example, using a set of heuristics, **softChange** regroups file revisions into MRs [5]. **softChange** analyzes the changes in the source code and thus extracts a list of function, methods and classes that have been added, modified or removed from one file revision to the next. **softChange** also correlates the available software trails; for example, **softChange** links a given MR to its Bugzilla bug report.
- **Visualizer:** **softChange** provides a visualizer to the repository that allows the user to explore the software trails. This front end is described in detail in the next section.

4. Visualizing software trails

One of the main purposes of **softChange** is to summarize and browse MRs. It will help developers, administrators and researchers explore and understand the development of the project. Instead of tedious typing, a developer or maintainer could quickly navigate through the MRs using the Web visualization front-end of **softChange**. The visualizer is divided in two main parts: a hypertext browser and a graphical viewer. The hypertext browser is used to navigate through the MRs. Users can choose to navigate MRs by date, by author, or by filename. For each MR, **softChange** provides the details of what revisions to which files it contains, and any metadata about the modification. The information is cross-referenced so it is possible to navigate amongst any related information by following hyperlinks.

softChange tries to leverage any external sources of information too. One benefit of the hypertext application is the ease of information association. Integration to other existing hypertext tools is quite easy by hyperlinking between tools. If the project provides a bugzilla repository (such as it is the case with many open source projects), a given MR is linked to its corresponding Bugzilla entry. **softChange** also links to the Bonsai repository of the project if one exists. Figure 4 shows a snapshot of **softChange** displaying the details of an MR for the Mozilla project.

The graphical viewer of **softChange** is composed of two main parts. One uses PostScript to generate static plots of the software trails. The other one uses SVG to display the same information more interactively. The SVG version takes advantage of its hypertext capabilities to link points in the plots with their details (by pointing to their details in the hypertext browser of **softChange**). **softChange** is able to generate the following plots:

- Growth of LOCS vs time, at the project level and at the module level (a module in **softChange** is defined as the collection of files under a given subdirectory).
- Number of MRs vs time: How many MRs are committed in a given period?
- Number of files vs time: How many files are part of the project at a given point in time?
- Number of files in a given MR: How many files compose a given MR?
- Proportion of MRs per contributor: What is the distribution of the number of MRs per contributor?
- Proportion of revisions per source code file: How frequently is a given file modified?



Project Mozilla



Details of Modification Request

[By Date](#) [By Author](#) [By File Name](#) [By Bugzilla Bug Number](#) [Search](#)

MR Id	Author	Files Modified	Date	Time	Description
mikep%oeone.com:2003/01/13 14:11:52	mikep%oeone.com	4	2003-01-13	14:11:52	Fixing bug 109476, in mc Fixing bug 188888, in mc Fixed thanks to patches

Files in MR

Filename	RevisionId	Lines Added	Lines Removed	Lines Total	State
calendar/resources/content/calendarEvent.js	1.45	27	0	27	Active
calendar/resources/content/calendar.xul	1.124	21	14	7	Active
calendar/resources/content/monthView.js	1.49	85	15	70	Active
calendar/resources/content/monthView.xul	1.19	12	10	2	Active
Total		4	145	39	106

Figure 2. Hypertext browser: details of an MR using softChange

- Number of modules that are modified in a given MR: How frequently an MR includes modifications of 2 or more modules?
- Project time-tree: When are given files created and modified, displayed in a timeline fashion?

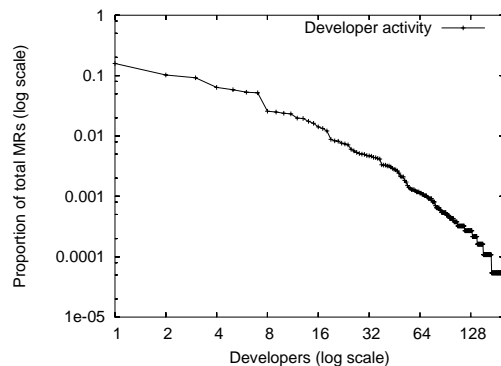


Figure 3. PostScript visualizer: proportion of MRs per contributor.

With the PostScript viewer, the user determines the necessary parameters for the plot, and softChange generates in return an encapsulated PostScript file. Figure 3 depicts the proportion of MRs per developer for Evolution (a GUI

mail client for Unix equivalent to Microsoft Outlook). Figure 4 depicts the number of MRs against time in the same project [6].

The SVG viewer takes advantage of the hypertext and interactivity features of SVG. This interaction is highlighted in the project time-tree diagram. The time tree graph is a view of a file directory tree. It depicts the how files populate a given directory (and its child subdirectories) and the proportion of MRs that include them at any point in time; the horizontal axis corresponds to time. Every time a file or directory is created, a new branch is started from the directory line. The user can expand and contract any given subdirectory, in order to avoid information overload. The user is also allowed to zoom-in, and zoom-out in any given region of the plot. Figure 5 depicts this diagram for the project Evolution.

5. Evaluation and Future Work

softChange has been successfully used to recover the history of the software project Evolution. The results are reported in [6]. softChange was used to extract Evolution's software trails, enhance them, and then query and visualize them. The Evolution project was born in 1999. By May 2003, its CVS repository kept track of almost 5000 files, for a total of 77,000 revisions. These revisions were reconstructed into 18,500 different MRs. A total of 201 developers committed at least one revision to the project.

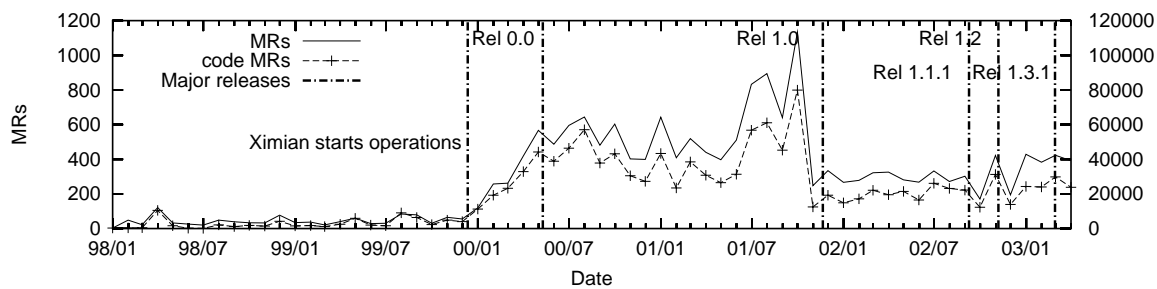


Figure 4. PostScript front-end: MRS over time.

Time-tree for the whole project

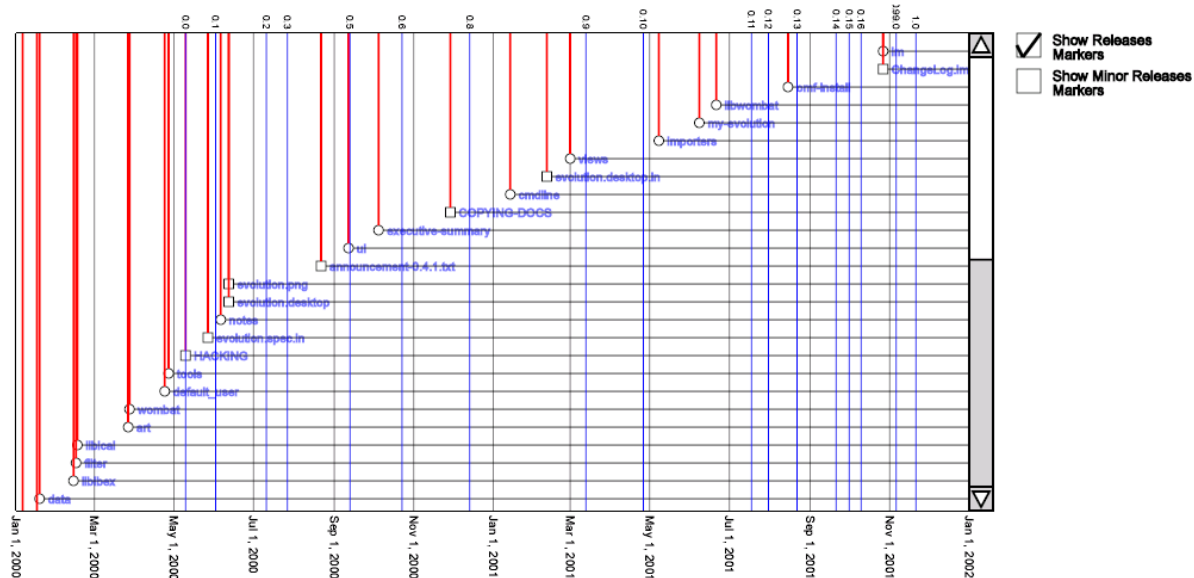


Figure 5. Time-tree in softChange

The size of the historical database created from the software trails of **Evolution** accounted for approximately 0.5 Gbytes. **softChange** helps us understand how the project evolved, and how its developers collaborated.

Another research project in which **softChange** was used is described in [4]. In this case, we were interested in understanding the way that the software developers of the GNOME project (a large, open source project) collaborate. The analysis of these software trails allowed the discovery of interesting facts about the history of the project: its growth, the interaction between its contributors, the frequency and size of the contributions, and the important milestones in its development.

Given that many of the plots and reports of **softChange** were designed around the questions described in the introduction, we are confident that **softChange** is useful to software developers and their management. We expect to maintain the historical data of several projects using **softChange** and make it available to developers, and then evaluate how they use it.

One of the main advantages of keeping all software trails in a relational database is that we can analyze them and enhance them by extracting new knowledge from them. Our current research is into the characterization of MRs. We want to know what types of MRs are typically committed by developers: are they source code modifications, documentation, or internationalization? If there are changes to the source code, are they bug fixes, new features, reorganization of the code or clean up? With this enhanced information, users can discriminate and select the changes they are interested in, without being overwhelmed by the amount of available data. The more facts that are known about the evolution of the project, the better the visualization tools that can be created.

Data mining of software trails is a promising area. Old, stable software projects have a large amount of software trails available. These trails can be mined for new facts; these facts can be used for better visualizations.

The architecture of **softChange** permits the use of different visualization tools. We are currently working with **JReflex** to adapt their Eclipse plug-in to **softChange**. We are also pursuing using **Shrimp** [10] to visualize the relationships available in the repository. **softChange** is an open source project, with an open architecture. We hope that other research projects will help create more trail extraction tools, more fact enhancing algorithms and more visualization tools.

Acknowledgments

This research was supported by the National Sciences and Engineering Research Council of Canada, and the Advanced Systems Institute of British Columbia.

References

- [1] D. Cubranic and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 2003 International Conference on Software Engineering*, pages 408–418, Portland, May 2003. Association for Computing Machinery.
- [2] M. Fischer, M. Pinzger, and H. Gall. Populating a Release History Database from Version Control and Bug Tracking Systems. In *Proceedings of the International Conference on Software Maintenance*, pages 23–32. IEEE Computer Society Press, September 2003.
- [3] M. Fisher and H. Gall. MDS-Views: Visualizing problem report data of large scale software using multidimensional scaling. In *Proceedings of the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA)*, September 2003.
- [4] D. M. German. Decentralized open source global software development, the gnome experience. *Journal of Software Process: Improvement and Practice*, Accepted for publication, 2004.
- [5] D. M. German. Mining CVS repositories, the **softChange** experience. In *First International Workshop in Mining Software Repositories*, 2004. To appear.
- [6] D. M. German. Using software trails to rebuild the evolution of software. *Journal of Software Maintenance and Evolution: Research and Practice*, To appear, 2004.
- [7] A. G. Gleditsch and P. K. Gjermshus. lrx Cross-Referencing Linux. <http://lrx.sourceforge.net/>, Visited Feb. 2004.
- [8] T. Hernandez. The Bonsai Project. <http://www.mozilla.org/projects/bonsai/>, Visited Feb. 2004.
- [9] Y. Liu and E. Stroulia. Reverse Engineering the Process of Small Novice Software Teams. In *Proc. 10th Working Conference on Reverse Engineering*, pages 102–112. IEEE Press, November 2003.
- [10] M.-A. D. Storey, C. Best, and J. Michaud. SHrIMP Views: An Interactive and Customizable Environment for Software Exploration. In *Proc. of International Workshop on Program Comprehension*, May 2001.
- [11] X. Wu. Visualization of version control information. Master’s thesis, University of Victoria, 2003.

A Framework for Comprehensive Experience-based Decision Support for Software Engineering Technology Selection

Andreas Jedlitschka, Dietmar Pfahl, Frank Bomarius
Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, D-67661 Kaiserslautern, Germany
[Andreas.Jedlitschka, Dietmar.Pfahl, Frank.Bomarius]@iese.fraunhofer.de

Abstract. *In today's software development organizations, methods and tools are employed that frequently lack sufficient evidence regarding their suitability, limits, qualities, costs, and associated risks. Supporting managers in selecting software engineering technologies that, on the one hand, have shown such evidence, and, on the other hand, match their business goal-oriented requirements, is the ambitious aim of software engineering decision support. This paper presents ongoing research towards a decision support system for software engineering technology selection in support for software process improvement management. After investigating the state-of-the-art, a framework for a decision support system is sketched and steps towards a decision support method are introduced.*

1. Introduction

It is widely accepted today that software development, similarly to software process improvement (SPI) has to be performed in a systematic and managed way. Unfortunately, at least for SPI, this is often not done. For example, it is reported that due to various shortcomings, SPI initiatives often fail. Debou et al. [1] describe three major causes for SPI failure: (1) lack of management commitment, reflected through too few or inappropriate resources, (2) delayed impact upon projects, both for daily practice and performance, and (3) lack of software management skills. Birk [2] reports that SPI initiatives often do not yield the expected results (level of improvement or benefit), since they are (1) performed in an isolated, non-coordinated way, (2) viewed outside their initial context, or (3) not supported by management or individuals.

Within SPI, decisions must be taken regarding software engineering techniques, methods, and tools that have a potential to yield significant improvements. To date, there have been few attempts reported that support SPI-managers in their task of decomposing strategic business objectives into activities on a SPI related

operational level. The question of which Software Engineering (SE) technique, method, or tool will fit best into the actual organizational context, is addressed either with support from external consultants, and/or based on the expertise and belief of the manager him-/herself. To find adequate answers, it is, beside other things, necessary to keep track of the state-of-the-art in SE, which is one of the basic "features" of a SE Decision Support System (DSS). SE DSS In general should implement the generation, evaluation, prioritization, and selection of solution alternatives [3].

In this paper, we describe a decision support framework that aims at enhancing SE technology selection in the context of improvement management. This framework has three major characteristics. Firstly, it offers decision support which is based on both internal and external experience. Typically, internal experience stems from project databases, measurement programs, and other company-specific organizational repositories. Typical sources for external experience are public web-based repositories, like the ESERNET repository [4][5]. Secondly, due to the extensive involvement of research organizations in the build-up and maintenance of public web-based repositories, framework users can expect to have access to state-of-the-art experience. Thirdly, the proposed framework will offer techniques and methods that help to aggregate and synthesize experience across the borders of different SE techniques along the software development lifecycle. More specifically, the framework will provide comprehensive decision support with regards to the application conditions of combinations of SE technologies, which is constructed upon pieces of information that stem from isolated empirical studies each focusing on a single SE technology.

A decision maker will benefit from easy access to the state-of-the-art for complementing his knowledge. Additionally, the DSS will provide directions for the most promising interplay of different techniques like inspections and testing, even if there are no concrete studies available that explicitly investigate the relationship between these two technologies.

The remainder of the paper is structured as follows. First, we present some related work in the different fields affected by this research (section 2). Next, we present the framework in a nutshell and outline the big picture (section 3). In section 4, we present steps towards a decision support method. In the final section, we summarize our contribution and outline plans for future work including a plan for validating the approach.

2. Related Work

We have identified three main areas affecting our work: (1) strategic management, which is most influential for all business-oriented activities but which is not in the focus of this paper, (2) decision support and decision support systems in general, for SE, and more specifically for improvement management, and (3) SPI and improvement management as application areas of the planned DSS.

2.1. Decision Support

Decisions have to be made every day by everyone. They are based on the experience, attitude, and intuition of the decision maker. They heavily depend on the information, the context, such as the budget and time, and other environmental restrictions or requirements. The complexity of decision-making grows with the impact a decision might have on the decision maker and the environment.

For a number of years, different research disciplines have looked at decision making in general (e.g., psychologists, system theorists). With the advent (and availability) of computers, more than thirty years ago, the interest in supporting decision making with information technology also arose. Power [7] proposes a matrix with five broad categories of DSS that differ in terms of technology component: communication-driven, data-driven, document-driven, knowledge-driven, and model-driven DSS. Additionally, he draws a distinction between user groups, purpose, and enabling technology.

For SE many different areas for decision-making are located along the life cycle of software. Ruhe [3] gives an overview on recent research related to SE Decision Support (SEDS). He describes five areas in which SEDS research is progressing. These are requirements, architecture and design, adaptive and corrective maintenance, project planning and control, and verification and validation.

For the area of systematic improvement, Birk [2] describes how to use technology experience packages (TEP) in support of technology selection. Based on a two-step context evaluation with respect to the given situation, appropriate TEPs are ranked and selected by the decision maker. The content of the TEPs can initially be acquired

from literature and iteratively be adapted to the specifics of an organization.

An integrated approach to simulation-based learning in support of strategic and project management is proposed by Pfahl [8]. Following a systems dynamics approach, he shows, that “based on simulations, managers can explore and analyze potential improvements before implementation and empirical evaluation of the related process changes in a pilot project”.

Raffo [9] suggests an approach to decision support using discrete-event simulation.

In the area of inspections, Biffel et al. [10] propose a knowledge management framework to support software inspection planning on three different managerial levels, quality management, inspection manager, and inspector.

2.2. Software Process Improvement

In contrast to the mass production of goods, software development projects are more or less unique. Consequently, improvement approaches valid in the production area are not appropriate without adaptation to the needs of software developing organizations. Generic frameworks include the Capability Maturity Model (CMM), ISO9000:2000, and the Software Process Improvement and Capability Determination (SPICE). These frameworks are standards for assessing software process maturity. They can be used for benchmarking against idealistic requirements. But they do not propose concrete techniques to be used in a specific context. The underlying improvement approaches are of a cyclic nature aiming at continuous improvement. The most prominent approaches here are the Quality Improvement Paradigm (QIP) [11], and SEI’s IDEAL Model [12].

3. The Framework in a Nutshell

Improvements, and especially the selection of appropriate SE techniques, have to be based on internal and external experience. No organization can afford to ignore existing experience or, in the worst case, make the same mistakes others made before. The framework we are focusing on is based on QIP and aims at motivating the extensive use of and need for empirical studies to support the selection of appropriate SE techniques within goal-oriented software process improvement management.

The suggested framework does not intend to replace existing reference models aimed at guiding organizations in improving their software processes, such as CMM, ISO9001, SPICE, or more recently CMMI, but provides a more generic view on the related topics

Figure 1 shows how organization-specific improvement management, including access to internal experience, is connected with external experience. In [13] we have described the motivation and requirements for

combining data sources from industry and research. Additionally, simulation could complement empirical work. The arrow from web-based DSS to industry indicates the usage of external data to enrich the organization-specific improvement management approach, especially for supporting decisions about technology selection. The arrows from and to research describe the contributions of, and benefits for, empirical research.

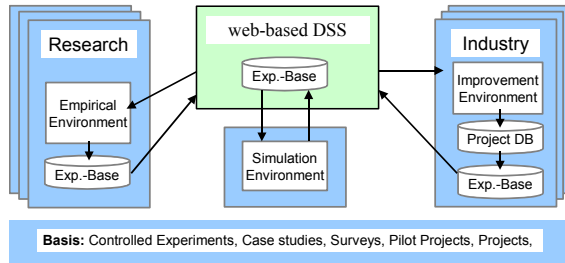


Figure 1. The big picture of SE-DSS

The arrow from industry to the web-based DSS is to a large extent still largely unresolved. Experience from SPI-focused network projects showed that industry is not sufficiently motivated to provide data in a reusable format [13], [14]. With few exceptions, only very abstract or even no information is reported.

4. Towards a Decision Support Method

To support decision-making, a model of the world has to be developed, e.g., [15]. At the moment, empirical software engineering research has a growing interest in analyzing results of different empirical studies with regard to the same technique, e.g., [16, 17, 18, 19]. Most of them also report challenges and threats occurring when studies are aggregated, e.g., because of limited information about the context.

Arranging the SE techniques alongside a virtual x-axis (the whole software lifecycle), we define the selection of studies investigating one technique (e.g., inspections) as local, and the selection of studies from different techniques as comprehensive. Deciding between different techniques requires at least some information about (1) the conditions that have to be fulfilled before the technique can be applied (precondition) and (2) the status that is available after having applied the technique (postcondition).

Figure 2 shows a sketch of an interplay graph for technologies via the pre-/post-condition interfaces. In this example, technology cluster X consists of inspection techniques and cluster X+1 consists of testing techniques. A virtual company (X-Soft) is using an object-oriented software development process, which results in UML diagrams at the end of the design phase. Following on

from the results of some empirical studies, an inspection technique specifically developed for object-oriented design documents is proposed (OORT - object-oriented reading technique). It is expected that with the introduction of OORT it might be realistic that 75% of the defects of type A+B and 50% of the defects of Type C will be found. Dynamic defects are not found at all.

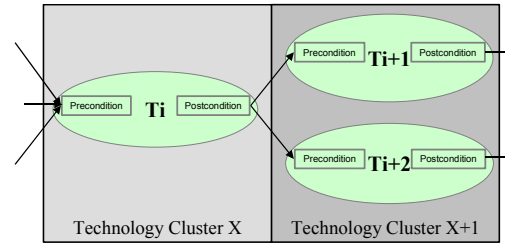


Figure 2. Interplay between technologies via pre/post conditions

The most interesting point is not the effect of a specific inspection technique as such but its impact on the whole development process (comprehensive view). So questions like: “Which inspection technique fits best with which testing technique?” are of growing interest. If it is possible to find empirical studies about testing techniques, using a similar or the same classification of defects, an answer to the question, under various restrictions, seems to be possible. This will not be of the same quality as if delivered by a statistical meta-analysis, but it will at least give a direction in areas where formal empirical evidence is still missing. Continuing the example, testing technique T_{i+1} is expected to detect 75% of type A+B defects and 60% of type C defects. T_{i+2} detects 50% of defect type A+B but 95% of defect type C. Both testing techniques detect 87% of the remaining dynamic defects. Considering only the testing technique as such, one might choose T_{i+1} , but in combination with the inspection technique T_i , it might be more promising to investigate in a combination of T_i and T_{i+2} , since from the empirical standpoint, they seem to be more complementary than T_i and T_{i+1} , and together they detect more defects than a single technique.

For inspection techniques, it is quite important to know when to apply which technique [10, 18], depending on the answer to questions such as: Is it more effective to inspect the requirement documents or design documents? Therefore, the arrangement of techniques in the development process plays an important role for which the DSS we aim at can provide directions.

5. Status of Work

After having clarified the vision for the DSS, we first conducted a literature survey to find the requirements for

a DSS as described above. Second, we did a survey among the software managers at Fraunhofer IESE to elicit further requirements. Together with additional requirements from the specification of the DSS we mapped our findings to a standard architecture and a generic requirements classification framework [20].

We have evaluated a set of controlled experiments in the area of defect reduction [6], especially towards their “usability” as “data” for a DSS. To get an idea of whether it is feasible to use the information available, and how to use it, we categorized the information needed for decision support and extracted such information from a small set of studies [21]. Based on this structure, we will determine the requirements for future reporting of studies, i.e., reporting schema, minimum attributes, and common measures to allow a comprehensive generation of information.

6. Summary and Future Work

In this paper, we have presented a vision for a decision support framework that supports the selection of appropriate SE techniques, by not only looking at a single technique in isolation, but also trying to aggregate information about the impact of the technique on other, related techniques in the software process at hand. A first step for a decision-support method was sketched, and a schema for the underlying repository has been presented.

There is further research necessary to evolve the vision into a running DSS. The architecture for the DSS and software design has to be fixed. It has to be evaluated whether the generic architecture given in [3] can be adapted. Assuming a simple three-tier-architecture with presentation, business logic, and persistence layer, the methods for the business logic, in particular, have to be further developed and evaluated.

Also, the evaluation of the whole approach has to be planned. At this time a two-step evaluation is foreseen. First, a controlled experiment with students as subjects will be used to get an idea of the efficiency and effectiveness of the approach. A survey among defect reduction experts and consultants should clarify whether, for example, the “base” is representative and whether the approach is able to complement state-of-the-practice with dynamic state-of-the-art information. We plan to report findings from these investigations in the near future.

Reference

- [1] Debou, C.; Kuntzmann-Combelle, A.: “Linking Software Process Improvement to Business Strategies: Experiences from Industry”, in *Software Process: Improvement and Practice* Vol 5, Number 1, March 2000, pp 55-64
- [2] Birk, A.: *A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering*; Ph. D. diss., Dept. of Computer Science, University of Kaiserslautern, Germany; Stuttgart: Fraunhofer IRB Verlag; 2000.
- [3] Ruhe, G.: “Software Engineering Decision Support – A new paradigm for Learning Software Organizations”. *Proc. Wkshp. Learning Software Organizations*, Springer, 2003.
- [4] Conradi, R.; Wang, A.I.: *Empirical Methods and Studies in Software Engineering – Experiences from ESERNET*; Springer LNCS 2765, 2003.
- [5] Jedlitschka, A.; Nick, M.: “Software Engineering Knowledge Repositories”; in [8] pp.55-80
- [6] Jedlitschka, A.; Ciolkowski, M.: “Towards Evidence in Software Engineering”; In *Proc. of ACM/IEEE ISESE 2004*, Redondo Beach, California, August 2004, IEEE CS, 2004
- [7] Power, D.J.: “A Brief History of Decision Support Systems”. DSSResources.COM, <http://DSSResources.COM/history/dsshistory.html>, version 2.8, May 31, 2003 visited 10.02.2003
- [8] Pfahl, D.: *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations*, Ph.D. diss., Dept. of Computer Science, University of Kaiserslautern, Germany; Fraunhofer IRB Verlag; 2001.
- [9] Raffo, D. M.: *Modeling Software Processes Quantitatively and Assessing the Impact of Potential Process Changes on Process Performance*, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, UMI Dissertation Services #963438, 1996.
- [10] Biffl, S.; Halling, M.: “A Knowledge Management Framework to Support Software Inspection Planning”, in [23]
- [11] Basili, V.R., Caldiera, G., and Rombach, H.D.: “Goal Question Metric Paradigm”; in: Marciniak JJ (ed.), *Encyclopedia of Software Engineering, Vol. 1*, pp. 528–532, John Wiley & Sons, 2001.
- [12] Gremba, J.; Myers, C.: “The IDEALSM Model: A Practical Guide for Improvement” in *Software Engineering Institute (SEI) publication*, Bridge, issue three, 1997. (<http://www.sei.cmu.edu/ideal/ideal.bridge.html>)
- [13] Jedlitschka, A.; Pfahl, D.: “Experience-Based Model-Driven Improvement Management with Combined Data Sources from Industry and Academia”. In: *Proc. of ACM/IEEE ISESE 2003*, Roman Castles, Italy, October 2003, IEEE CS, 2003. pp. 154-161
- [14] Conradi, R.; Dybå, T.; Sjøberg, D.; Ulsund, T.: “Lessons Learned and Recommendations from two Large Norwegian SPI Programmes”. In Oquendo (ed.): *9th Europ. Wkshp. on Software Process Technology (EWSPT 2003)*, Helsinki, Finland, 1-2 Sept. 2003, pp. 32-45. LNCS 2786, Springer-Verlag.
- [15] Ruhe, G.: “Software Engineering Decision Support: Methodology and Application”s. In: *Innovations in Decision Support Systems* (Ed. by Tonfoni and Jain), International Series on Advanced Intelligence Volume 3, 2003, pp 143-174.
- [16] Pickard, L.M.; Kitchenham, B.A.; Jones, P.W.: “Combining empirical results in software engineering”, *Information and Software Technology*, 40(14):811-821, 1998.
- [17] Runeson, P.; Thelin, T.: “Prospects and Limitations for Cross-Study Analyses – A Study on an Experiment Series”. In [22] pp. 136-145
- [18] Wohlin, C.; Petersson, H.; Aurum, A.: “Combining Data from reading Experiments in Software Inspections”. In: *Lecture Notes on Empirical Software Engineering*, World Scientific Publishing Co. Pte. Ltd, Singapore, 2000.
- [19] Kitchenham, B.A.; Dybå, T.; Jørgensen, M.: “Evidence-based Software Engineering”, accepted for ICSE 2004
- [20] Jedlitschka, A.; Pfahl, D.: “Requirements of a Tool supporting decision making for SE Technology Selection”; In *Proc. of SEDECS 2004 Workshop at SEKE 2004*, Banff, Canada, 2004
- [21] Jedlitschka, A.; “Towards a comprehensive summarization of empirical studies in defect reduction”; IESE Report 043.04/E, submitted as “Fast Abstract” to ISESE 2004
- [22] Jedlitschka, A.; Ciolkowski, M. (eds): *The Future of Empirical Studies in Software Engineering*, Proc. of 2nd Int. Wkshp. on Empirical Software Engineering, WSESE 2003, Roman Castles, Italy, Sept. 2003, Fraunhofer IRB Verlag, 2004, to appear.
- [23] Aurum, A.; Jeffery, R.; Wohlin, C.; Handzic, M. (Eds): *Managing Software Engineering Knowledge*; Springer-Verlag; Berlin 2003

Active Connectors for Component-Object based Software Architecture

Tahar Khammaci, Adel Smeda, and Mourad Oussalah

LINA, Université de Nantes

2, Rue de la Houssinière, BP 92208

44322 Nantes Cedex 03, France

Tel: +332 51125963, Fax: +332 51125812

Email: {khammaci, smeda, oussalah}@lina.univ-nantes.fr

Abstract. Component-Object based Software Architecture (COSA) describes systems as a collection of components that interact with each other via connectors, it separates the notion of computation from the notion of communication. However connectors in COSA are passive, in the sense that they only provide interaction services in response to explicit requests; hence they do not have the ability to react to situations that arise during the process, in consequence they cannot react to changes in components and configurations. In this article we present special purpose connectors based on Event-Condition-Action rule called active connectors.

1. Introduction

There are a number of circumstances that can arise in component-based systems. This includes changes of a system configuration, up normal termination of interaction process due to some reasons, changes in components, etc. For a system to survive, it must react to these situations (events). Traditionally, connectors are responsible only for interacting components with each other. Hence they are passive, in the sense that they do not react to events that could take place in a system.

Active connectors are intended to respond automatically to situations that arise inside a system. The active behavior of a connector is described using rules, which most commonly have three statements, an event, a condition, and an action. A rule with such statements is known as an even-condition-action rule or ECA rule [1]. Such a rules lies dormant until the occurrence of the event that it is monitoring. When the event takes place, the condition is evaluated to examine the context in which the event took place, and if true the action is executed to enact some suitable response to the event.

The concept of ECA is clearly not new, it is supported within a rang of proposals for rule systems (e.g. OODBS [2], [3], [4], [5]). However, we are not aware of any other attempt to apply the ECA concepts in component-based software architecture.

2. Connectors in COSA

COSA is an approach of describing software architecture based on components, connectors, and configurations. Where components, connectors, and configurations are defined as types that can be instantiated to build different architectures [6]. On the contrary to most of architecture description approaches [7] [8] COSA considers connectors as first-class entities that must be defined explicitly by separating their interfaces from their behavior [9]. A connector in COSA is mainly represented by an *interface* and a *glue* specification. In principle, the *interface* shows the necessary information about a connector, including number of *roles*, service type that the connector provides (communication, conversion, coordination, facilitations), connection mode (synchronous, asynchronous), transfer mode (parallel, serial), whether it includes ECA rule or not and if yes the type of events that supports, etc. The interaction points of an interface is called *role*. A *role* is the interface of a connector intended to be tied to a component interface (a component's port). In the context of the frame, a *role* is either a *provide role* or a *require role*. A *provide role* serves as an entry point to a component interaction represented by a connector type instance and it is intended to be connected to the *require* interface of a component (or to the *require role* of another connector). Similarly, a *require role* serves as the outlet point of a component interaction represented by a connector type instance and it is intended to be connected to the *provide* interface of a component (or to the *provide role* of another connector). The interface is the visible part of a connector, hence it must contain enough information regarding the service and the type of this connector. By doing this, one can decide whether or not a given connector suits its qualifications by examining its interface only.

The *glue* specification describes the functionality that is expected from a connector. It represents the hidden part of a connector. A *glue* could be just a simple protocol links the roles or it could be a complex protocol that does

various operations including linking, conversion of data format, transferring, adapting, etc. The service provided by a connector is defined by its *glue*, the services of a connector could be either communication service, conversion service, coordination service, or facilitations service. In case of a composite connector the subconnectors and subcomponents of the composite connector must be defined by the *glue*, as well as the *binding* among the subconnectors and subcomponents.

3. Active connectors

The association between situations and actions is specified by means of rules. A rule in the sense of active connectors is a situation-action pair; a situation is generally specified by means of an event and a condition, where an event indicates an occurrence in the architecture and a condition relates to the current state and can be formatted as a predicate over it. A condition has to be evaluated when the corresponding event is signaled; if it holds, the associated action has to be executed. The overall structure of rule definitions for active connectors is expressed as follows, the keyword “Active” is presented to activate or deactivate the rule:

Mode \subset {immediate, deferred, separate }; }

In active connectors events to trigger the rules are indispensable, therefore the rule CA is not applicable here. Meanwhile conditions can be omitted, hence the rule EA is applicable for active connectors.

Figure 1 elucidates the definition of active connectors using a UML meta-model. In the figure the connector includes the rule, which is triggered by the event. In component-based systems there are two types of events: component events and interaction events. The rule includes a condition, which could be omitted, and an action to be taken if the condition is satisfied.

3.1. Basic events for active connectors

Each event specification system must start with a list of basic events that the system supports. While this set of basic events could vary from system to another, there are some basic events that we consider important in the context of component-based systems and component-based software architecture. We classified the events for active connectors in two groups, events concerning with components and events concerning with interaction process.

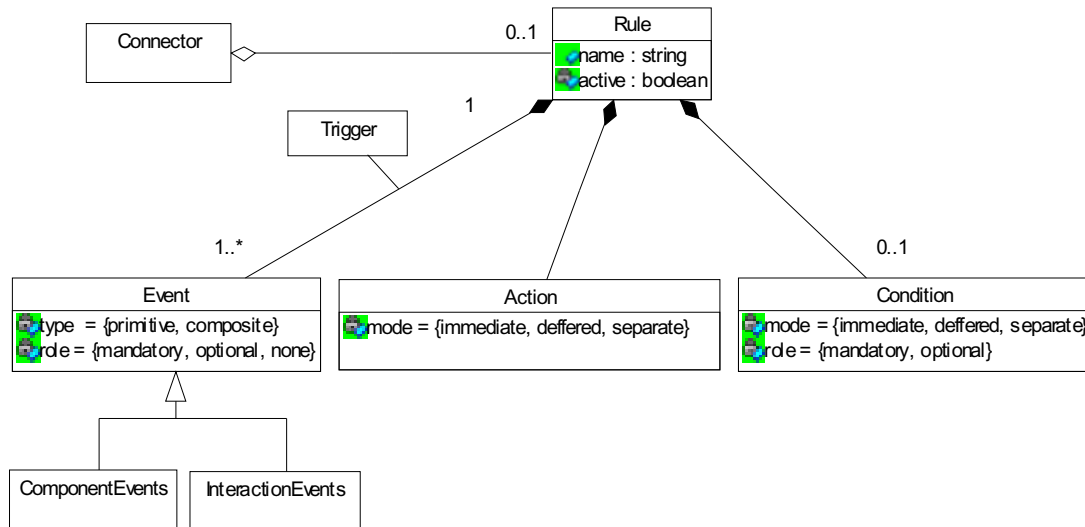


Figure 1. A UML meta-model of active connectors in component-based system

```

DefineRule rule-name {
    Active  $\in$  {Yes, No}
    ON { define-event (or events);
        Type  $\subset$  {primitive, composite}; }
    IF { define-condition;
        Mode  $\subset$  {immediate, deferred, separate };
        Role  $\in$  {mandatory, optional}; }
    DO { define-action;

```

1- Component events

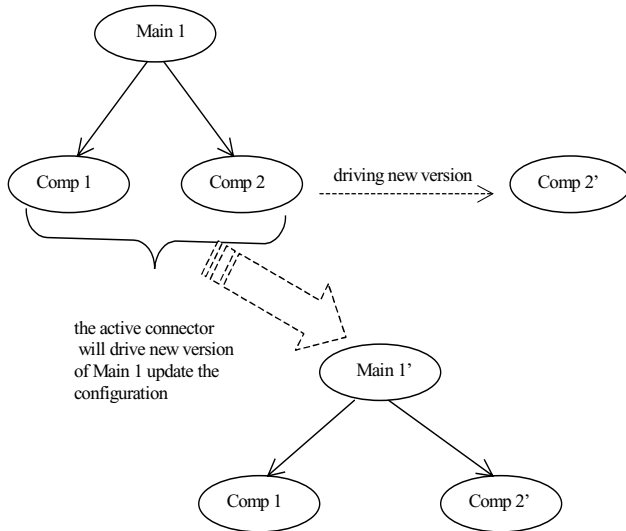
Since the target of the model we are seeking is a dynamic system configuration, the model should provide us with the ability to describe arbitrary changes in configurations of components and component assemblies. In particular the model should allow for the dynamic creation of new components and new component assemblies, as well as

the movement of components or component assemblies from one component assembly to another. In other terms, containment and resource dependencies among components should be allowed to change and evolve over time, whether spontaneously or in reaction to interactions among components.

Thus an obvious connector's task is dynamic component linking that allows for a variety of dynamic changes to the system architecture and deployment (e.g., component creation, deletion, migration, replacement). In this case active connectors are needed to support changes and to repair existing systems (configuration changes). Components' events occur when such situations take place in a configuration.

Conventional connectors can not react to configuration changes, therefore new design must be considered. In the other hand, active connectors facilitate changes in a system by reconfiguring the system without a direct interfere from the components. Component events take place when configuration changes occur, for instance, immediately after a component is created, immediately before a component is deleted, or immediately after a new version of a component is driven (e.g. in a composite relation). Therefore, with component events a rule must be checked every time component changes take place, i.e. creation, deletion, migration, replacement.

Example: In a composite component, driving a new version of a component which depends on an other component needs that the independent component also must be driven. In this case the connector will drive a new version of the dependent component and update the configuration (figure 2). In the figure, after component Comp2' is driven the connector will drive a new version of component Main 1 and updates the relations.



```

DefineRule change-propagation {
    Active = Yes

```

```

ON { drive-version of component ;
    Type = primitive; }
IF { dependence = ON;
    Mode = immediate;
    Role = mandatory }
DO { new component = component.drive;
    new component.delete component ;
    new component.add component;
    Mode =immediate; }
}

```

Figure 2. Using an active connector to drive versions in a composite relation.

2- Interaction events

Events that arise to control an interaction process are called interaction events. They are events that used to supervisor interactions and communications among components. They are intended to prevent communication problems and to facilitate interactions in a system. There are many events that can be seen as interaction events including, communication events, facilitation events, and coordination events.

1- *Time events* are examples of coordination events; they occur periodically to predict and prevent communication errors. Time events are specified as: at *time-specification*, every *time-period*, after *time-period*.

Example: The following rule says that after certain time a connector will stop the connection if the receiver does not respond to the request.

```

DefineRule connection-postpone {
    Active = Yes;
    ON { time-period (MIN=15);
        Type = primitive;}
    IF { receiver = !ready;
        Mode = immediate;
        Role = mandatory; }
    DO { stop connection;
        Mode = immediate; }
}

```

2- *Communication events* are used to support and to guarantee data transfer among components, consequently they are triggered when a transfer takes place. Communication events could occur:

- Immediately after a transfer begins (tbegin).
- Immediately after a transfer finishes (tfinish).
- Immediately before a transfer aborts (tabort).
- Immediately after a transfer aborts (tabort).

Example. A connector sends an acknowledgment to a sender and a receiver engaged in communication says that

the transfer of the data is terminated successfully, this can be realized by the following rule:

```
DefineRule transaction-terminate {
    Active = Yes;
    ON { after tabort;
        Type = primitive; }
    IF {transaction = success;
        Mode = immediate;
        Role = mandatory; }
    DO {send ACK;
        Mode = immediate; }
}
```

3- *Facilitation events.* To enable heterogeneous components facilitation services such as load balancing and format conversion are required. Number of events can be considered as facilitation events, this includes format mismatching, synchronies mismatching, receiver overflow, traffic congestion. For instance a rule could define a load balancing mechanism when a congestion occurs by switching the interaction to other connectors, if possible, after studying the root (from the source to the destination).

Example: The following rule defines what must be done when a receiver overflow occurs. The rule is an event-action type since there is no condition.

```
DefineRule receiver-overflow {
    Active = Yes;
    ON { overflow = 1;
        Type = primitive; }
    DO { stop transaction;
        while receiver = !ready
            wait ;
        Mode = immediate; }
}
```

With interaction events a rule must be checked every time a communication service is needed, e.g. data access, data and messaging exchange, broadcasting etc.

4. Conclusion

In this article we investigate the application of active rules for the connectors of component-object based software architecture (COSA). We can conclude that the inclusion of powerful active mechanisms into component-based software architecture provides powerful services, such as reacting to situations and external notifications, reacting at specific points in time, management of (consistency) constraints, access control, or automatic propagation of updates (architecture changes). Furthermore, using active connectors increases the flexibility of systems in that e.g. composite relation updates, inherited models updates.

Most of these features are not supported by conventional component-based systems now.

References

- [1] U. Dayal, A. Buchmman, and D. McCarthy, "Rules are Objects too: a Knowledge Model fro an Active Object-Oriented Database Systems", *Lecture Notes in Computer Science 334*, Springer, 1988, pp. 129-143.
- [2] S. Gatziau, A. Geppert, and K. Dittirch, "Integrating Active Concepts into an Object-Oriented Database Systems", *In Proceeding of DBPL-3 Workshop*, Nafplion, Greece, 1991.
- [3] N. Paton, O. Diaz, M. Williams, J. Campin, A. Dinn, and A. Jaime, " Dimensions of Active Behaviour" , *In Proceedings of the 1st International workshop on Rules in Database Systems*, Edinburgh, Scotland, 1994, pp. 40-57.
- [4] D. McCarthy and U. Dayal, "The Architecture of An Active Data Base Management System", *In proceedings of the ACM SIGMOD symp. on the Management of Data*, Portland, Oregon, 1989, pp. 215-224.
- [5] N. Gehani, H. Jagadish, and O. Shmueli, "Event Specification in an Active Object-Oriented Database", *In Proceedings of the ACM SIGMOD International Conference*, Dan Diego, CA, 1992, pp. 81-90.
- [6] A. Smeda,, M. Oussalah, and T. Khammaci, "A Multi-Paradigm Approach to Describe Software Systems", *In Proceedings of 3rd WSEAS Int. Conf. On Software Engineering, Parallel and Distributed Systems*, Salzburg, Austria, 2004.
- [7] L. Bass, P. Clements, and R. Kazman, R. "Software Architecture in Practice", Addison-Wesley, Indianapolis, IN, 1998.
- [8] N. Medvidovic, R. N. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE Transactions on Software Engineering*, Vol. 26, 2000, pp. 39-70.
- [9] M. Oussalah, A. Smeda,, and T. Khammaci, "An Explicit Definition of Connectors for Component-Based Software Architecture", *In Proceedings of the 11th IEEE Conference on Engineering of Computer Based Systems (ECBS 2004)*, Brno, Czech Republic, May 24-27, 2004.

Analyzing Invariant Condition of Running Java Program

Theodorus Eric Setiadi¹, Ken Nakayama, Yoshitake Kobayashi, and Mamoru Maekawa

Department of Information Systems

The Graduate School of Information Systems

The University of Electro-Communications, Tokyo Japan

{eric| ken | yoshi| maekawa} @maekawa.is.uec.ac.jp

Abstract. Abstraction is used to simplify understanding of a complex system. Abstract thinking is easy for human, but unfortunately making an abstract model for a real system is not easy. This paper describes an interactive environment called Java Program Analyzer (Javapan) that helps abstraction process. Javapan helps identifying the partial specification that can be used as the input for the verification system. Using some clues from source code and statistical analysis of the execution trace, Javapan reports candidates of invariant conditions that consistently hold during the execution. The user can supervise the analysis by giving some particular conditions to be observed. The user also can inspect the execution by viewing the property for specific part of program. A preliminary example is included to illustrate how to obtain the invariant conditions from a running Java program using Javapan.

1. Introduction

Abstraction makes an actual complex system becomes simpler. For human, given an abstract model, it is easy to think abstractly. Abstraction is also useful in verifying, testing, debugging, and maintaining a system. Verification tools (for example SPIN [1], SMV[2] model checker) are powerful [3], but there is a need to have an abstract model so it could be practically feasible for the actual problem. Testing and debugging a complex program is difficult task for programmer. Sometimes it requires a lot of effort, such as finding a good test data, analyzing a lot of result, and reading lots of codes. Testing, modifying, and understanding program accounts for a half of the development time [4]. Abstraction can be helpful in software engineering, by some abstraction model or property of the program in hand, testing and debugging a program becomes easier. For example, in the Print Service [5] problem, it would be helpful for the user if he knows that each request for a printer will eventually get processed. By knowing this, the user will be able to

concentrate on that property during the testing and debugging.

Program visualization also expresses the need for abstraction. These tools give abstraction of the actual system in visual representation by providing a way to view the execution, *e.g.*, viewing the value of variable or displaying the statistic from the trace [6], [7].

Unfortunately, it is difficult to analyze the target problem and make an abstract model. This makes additional work for the user. Moreover, typical usage or typical data of the system needs to be known. Good documentation is also necessary. Historically, programmers have been reluctant for writing formal specifications [4]. Looking at this situation, there is a need to have a system that could help the abstraction process, for example gives some assertions about the program execution.

This paper describes a Java Program Analyzer (Javapan) that helps abstraction. More specifically, Javapan analyzes the behavioural property of a running Java program and reports invariant conditions that consistently hold during the program execution. Those are the conditions that are always true or never exist during the execution. The users of Javapan are programmers who need help to understand a program. This system interacts with the user to find some candidate assertions, by allowing the user to supervise the analysis, for example, the user can inform the system to analyze a particular expression or view the property for some part of the program. The output from Javapan can be used to help in writing partial specification for the program that possibly can be used as the input for the verification system.

2. Approach Overview

Javapan is implemented as an Eclipse plug-in [8]. Javapan reads Java source code and receives the execution trace from the Java Development Tool (JDT) plug-in. It analyzes the program behaviour from the execution trace using some clues from the source code to find useful property that reflects the intention of the programmer.

¹ The author is supported by the JINNAI scholarship.

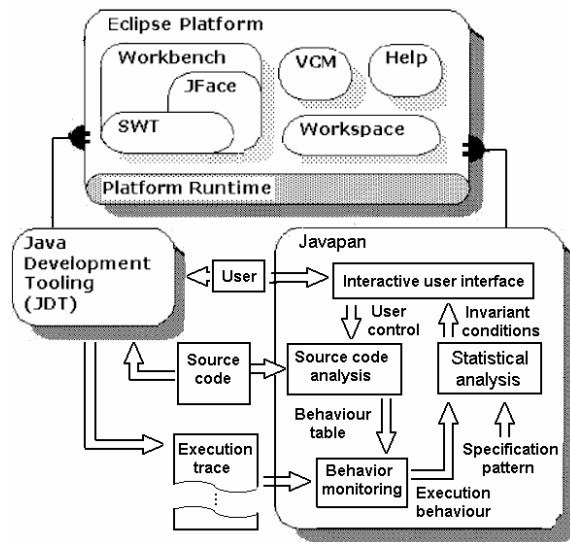


Figure 1. System architecture of Javapan

3. Proposed Methodology

3.1. Source Code Analysis

In source code analysis, Javapan parses the Java source code to find some expressions. In Java source code, an expression is a statement that can convey a value [11]. The most common expressions are mathematical expressions (such as $x=3$). They can be found in the assignment, if statement, while statement, *etc.* From the observation of some program examples (such as Print Service [5], Readers Writers [9], Expandable Array [9], Bounded Buffer [10] problem, *etc.*) usually the expressions stated in the source code are useful to find the property of the program. To illustrate how the Javapan analyzes the source code, let's take a look at an example of a Print Service problem [5] (Figure 1). In this system a set of Print Service objects organized in a ring topology, passing around rights to access a Printer. In case one node does not poses the printer, it asks its neighbour for it.

```

1: class PrintService {
2:   protected PrintService neighbor = null;
3:   protected Printer printer = null;
4:   public synchronized void print(byte[] doc) {
5:     getPrinter().printDocument(doc);
6:     return; }
7:   protected Printer getPrinter() {
8:     if (printer == null)
9:       printer = neighbor.takePrinter();
10:    return printer; }
11:   synchronized Printer takePrinter() {
12:     if (printer != null) {

```

```

13:       Printer p = printer;
14:       printer = null;
15:       return p; }
16:   else
17:     return neighbor.takePrinter(); }
18:   synchronized void setNeighbor(PrintService n) {
19:     neighbor = n; }
20:   synchronized void givePrinter(Printer p) {
21:     printer = p; }
22:   }

```

Figure 2. Part of source code of the Print Service problem

In Javapan, the expressions that are obtained from the source code are called “main expressions.”

Table 1. Some of the main expressions for the source code in Figure 2

Main expressions	Type and location
neighbor=null	Assignment (line 2)
printer=null	Assignment (line 3)
printer==null	If statement (line 8)
...	...

Javapan also analyzes some “related expressions” of the main expressions. For example $neighbor \neq null$ is related to the $neighbor == null$. Javapan then replaces the assignment operator ($=$) with comparison operator for equality ($==$) so the expressions only contains comparison operator ($==$, $!=$, $<$, $>$, $<=$, $>=$). Figure 3 is the summary of the main expressions and related expressions that only using comparison operator for the source code in the Figure 2.

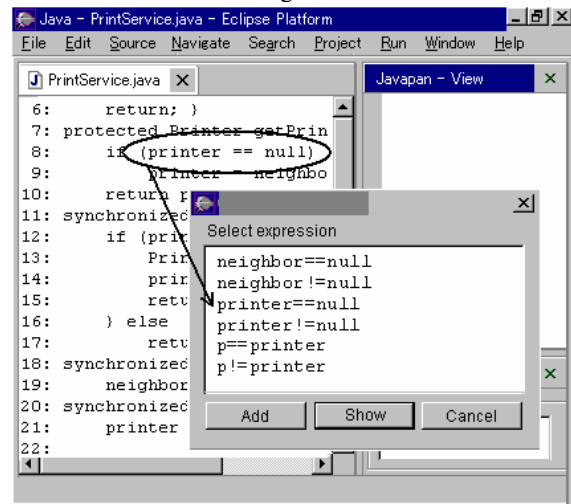


Figure 3. Expression for the analysis

User can add some expressions that have not been listed by choosing the add button in Figure 3.

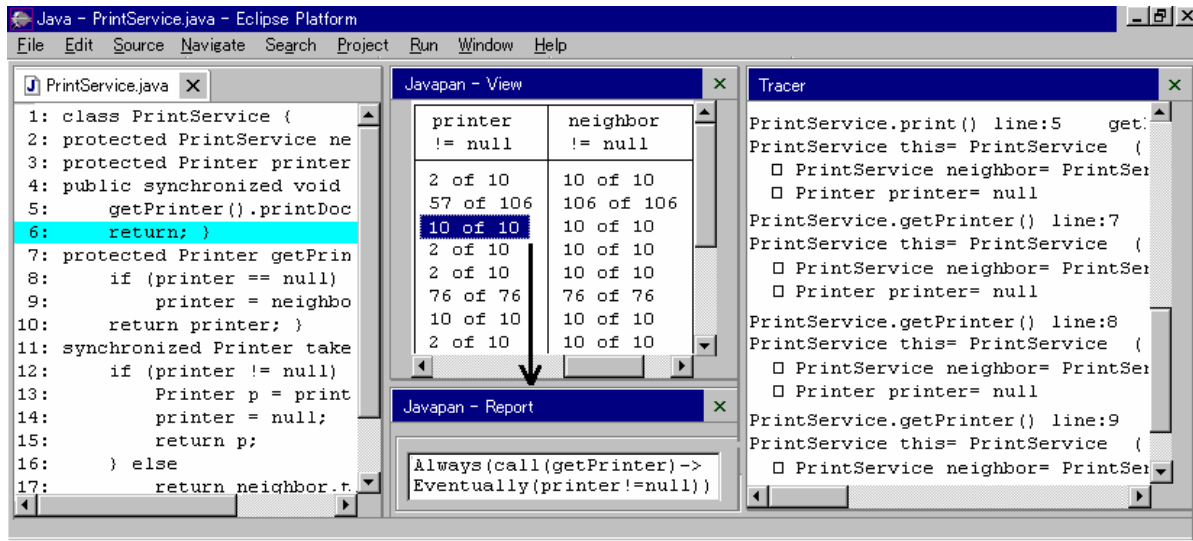


Figure 4. Javapan view and report

3.2. Tracing

Javapan monitors the program behaviour from the execution trace. Execution trace contains information about class name, method name, value of variable, *etc* (see the Tracer in Figure 4). Javapan obtains the execution trace from the Java Development Tool's debugger of Eclipse. It puts some breakpoints in the source code at the some points where exists statements such as entry of a function, assignment, return *etc*. For the analysis, Javapan creates behaviour table that records the valuation of the expressions from the source code analysis and their related expressions. It records how many times the expression occurred as true and also the number of times the line of code have been executed. Javapan - View in Figure 4 shows the behaviour table for the Print Service problem.

3.3. Statistical Analysis

Javapan uses specification pattern [12] to express the property of the program. It is a collection of patterns about property specification that occur commonly in the specification of concurrent and reactive systems. The invariant conditions in Javapan can be grouped into these 3 patterns [13]:

- **Always:** the condition that is always holds as true. For example Expandable Array [9], the actual number of elements never gets greater than the array capacity. `Always(element <= array.capacity)`
- **Absence:** the condition that never exists. For example Readers Writers problem [9], the Readers Writers

exclusive access property states that simultaneous reading and writing is not admitted. Never (`activeReaders > 0 && activeWriters > 0`)

- **Response:** cause-effect relationship between a pair or events/conditions. An occurrence of the first must be follow by an occurrence of the second. `Always(cause -> Eventually(effect))`

In the behaviour table, the expression with the occurrence 0 of n indicates the absence pattern and the one with occurrence n of n (n is not 0) indicates always pattern. Looking at the Javapan - View in the Figure 4, there are some invariant conditions within the program execution. When the `getPrinter()` method was called (line 5) sometimes the value of the variable printer was null (`printer == null`) and sometimes was not null (`printer != null`), but at the return of the method (line 6), the value of the printer variable was always not null (`printer != null`) with the occurrence 10 of 10. Since the return will follow the function call, therefore Javapan reports that

`Always(call(getPrinter()) ->
Eventually(printer != null))`

It means that every print service will eventually get the printer upon request (see the Javapan - Report in the Figure 4). Javapan reports the property is in temporal logic notation.

Another invariant condition that is found in the Print Service example is `Always(neighbour != null)` because after the `startUpServices()`, the `PrintService` will have a neighbour. It can be seen from the Javapan - View in Figure 4 that the occurrence is always n of n.

4. Discussion

Since this system analyzes the behaviour from execution trace, the output is dependent from the input. Figure 4 shows an example of behaviour of 5 Print Services with 10 requests. If the requested PrintService doesn't have the printer, it will ask its neighbour so that line 9: `printer = neighbor.takePrinter();` will be executed. It can be seen from the behaviour table that the occurrence of the condition `printer != null` is 76 of 76. If the input data only requests to 1 Print Service which already has the printer, then the line of code 9 will never be executed. Therefore, the occurrence of `printer!=null` in line 9 would be 0 of 0 and this system report that at this point, that condition is never occurred. This fact shows that the input data is an important factor in this system.

5. Related Work

Ernst *et al.* [15] also has proposed a dynamic discovery of invariants from variable traces. It uses a set of possible invariants to be tested. Our work differs in that Javapan uses some useful clues from the source code to guide the analysis.

Glenn *et al.* [4] has proposed a system to mine specifications using the execution trace. The output is specification in the form of an automaton, which is generated from the traces. It requires further analysis from the user to understand what the resulting automaton implies, whereas Javapan gives the property of the program stated in temporal logic. This enables our system to get the important property explicitly.

6. Conclusion and Future Work

In this paper, we discussed a tool that helps abstraction called Javapan. Using the clue from source code and statistical analysis of the execution trace, Javapan reports invariant conditions that consistently hold during the program execution. The output from Javapan can be used to help writing the partial specification of a program and possibly can be used as the input for the verification system. We also showed a preliminary example to illustrate how Javapan can help in finding invariant conditions in the program execution.

In the future, we plan to extend this system so that not only invariant conditions are reported, but also this system can detect anomaly of program behaviour. The work from Michael and Gosh [14], uses learning finite automata to identify specific program behaviour. Given a

normal behaviour, this learning finite automaton can also be used to detect the anomaly, but it requires the user to give some learning data of the normal behaviour. By statistically analysing the execution trace, we hope it will be able to distinguish the anomaly from the normal behaviour.

References

- [1] Gerard J. Holzmann. "The Spin Model Checker: Primer and Reference Manual." Addison-Wesley. ISBN 0-321-22862-6. September 2003.
- [2] Kenneth L. and McMillan. "Symbolic Model Checking." Kluwer Academic Publishers, 1993.
- [3] Edmund M. Clarke and Jeannette M. Wing. "Formal Methods: State of the Art and Future Directions." ACM Computing Surveys. 1996.
- [4] Glenn Ammons, Rastislav Bodik, James R. Larus. "Mining Specifications." Symposium on Principles of Programming Languages. 2002.
- [5] Doug Lea. "Concurrent Programming in Java. Addison-Wesley 2000." pp 113 - 114
- [6] Johan Moe and David A. Carr, "Understanding Distributed Systems via Execution Trace Data". 2001
<http://citeseer.nj.nec.com/moe01understanding.html>
- [7] Javix Program Analyzer. JAVIX CORPORATION All rights reserved. 2000. <http://www.javix.com/>
- [8] IBM Corp. "Platform Plug-in Developer Guide." 2001. <http://www.eclipse.org/>
- [9] Radu Iosif and Riccardo Sisto. "YAV: A Formal Verification Software for Java: Case-Studies." <http://www.dai-arc.polito.it/dai-arc/manual/tools/yav/case-studies/>
- [10] Klaus Havelund. Java Pathfinder User Guide. NASA Ames Research Center, Recom Technologies, Moffett Field, CA, USA. August 1999.
- [11] Laura Lemay and Rogers Cadenhead. "Sams Teach Yourself Java 2 in 21 Days (Teach Yourself in 21 Days Series)." SAMS Publisher. April 1999.vv
- [12] Matthew B. Dwyer, George S. Avrunin and James C. Corbett. "Property Specification Patterns for Finite-state Verification." The 2nd Workshop on Formal Methods in Software Practice, March, 1998.
<http://www.cis.ksu.edu/~dwyer/papers/spatterns.ps>
- [13] Matthew B. Dwyer, George S. Avrunin and James C. Corbett. "Patterns in Property Specifications for Finite-state Verification." Proceedings of the 21st International Conference on Software Engineering, May, 1999.
- [14] Christoph Michael and Anup Ghosh. "Using Finite Automata to Mine Execution Data for Intrusion Detection: a Preliminary Report." Lecture Notes in Computer Science. Vol 1907 p66. 2000.
- [15] Michael D. Ernst, Jake Cockrell, William G. Griswold, David Notkin, "Dynamically Discovering Likely Program Invariants to Support Program Evolution." International Conference on Software Engineering. p213-224. 1999.

Application Semiotics Engineering Process

Gang Zhao

STARLab, Department of Computer Science, Vrije Universiteit Brussel, Belgium
gang.zhao@vub.ac.be

Abstract. As application semantics becomes more complex and dynamic in IT systems, it is necessary to engineer the application semantics in its own lifecycle of development parallel to system engineering. The application semiotics engineering process is under study as a methodology for engineering complex and dynamic business logic in intelligent applications. It stresses the informal specification, the traceability of engineering decisions and need of late-binding to a particular formal language representation and computational paradigm for distributed multidisciplinary collaborative modelling environment. The article describes how the application semiotics is developed in a lifecycle of iterative development.

1. Introduction

As IT system requirements modelling goes beyond data and operational paradigms to the underlying business rationale, there arises the need to explicitly capture the business semantics and deploy it in a system, encapsulated in response to its dynamic changes in business model, process and rationale. The explicit model of business semantics is the corporal knowledge and important parts of software assets.

This paper presents an on-going exploration of an engineering process to model such semantics and how its activity is best organized on the insight from database modeling, knowledge system development, object-oriented and component-based software engineering, domain engineering and ontology research. It first introduces what the application semiotics engineering process (ASEP) is and describes an ontology-based approach to application semiotics. It illustrates the lifecycle and activities of the process and key instruments and approaches of the methodology.

2. Application semiotics engineering process

Semiotics is a science of signs and their interpretation [2, 3, 4, 22]. Here it is used to refer to a system of signs. A semiotic system is a model of human intelligence or knowledge or logic for communication or cognition. As a

semiotic, it has three aspects: semantic, syntactic and pragmatic [21, 28]. It presents the conceptualization of ‘subject world’ [13] in well-formed symbolics and specifies how it is interpreted and processed with respect to specific application contexts.

Semiotics engineering is a process of creating a symbolic system. It is similar to the development of computational models of data, process, object, knowledge or ontology. It includes such tasks as scoping, modeling, integration, deployment and maintenance. It takes two fundamental viewpoints of semiotics: the *capture* and *use* of application semiotics. The capture seeks for semantic *presentation* for communication and consensus. It operates on the semantic and syntactic dimensions of semiotics. It is often concerned with model scalability and reusability. The use emphasises the formal *representation* for processing and reasoning. It is concerned with pragmatics of semiotic models: from which perspective and in what context of application the semiotic model is applied with computational consistency and effectiveness.

Intelligent information systems can be plotted along the capture and use dimensions in terms of *specificity* and *diversity*:

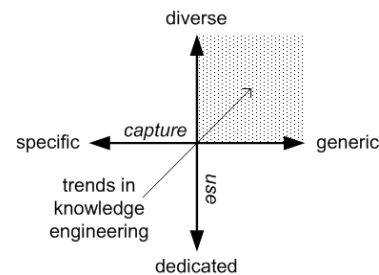


Figure 1. Specificity of capture and diversity of use

The development of intelligent information systems aspires to move from domain specific semiotics and their dedicated use towards domain generic model in versatile applications. The ASEP is intended for modeling complex business rules, application logic and domain knowledge which need either encapsulated for change or separated from conventional software modelling of functional dimensions of IT systems for different development or asset management. It targets systems with rich application semantics, such as knowledge systems, system integration

with divergent and rapidly changing business logic, semantic interface specification of software components or web services, protocols for semantic interoperation of collaborative processes or systems. The ASEP is aimed at the development of corporate or organizational intelligent systems and open services such as knowledge management systems, semantic web services [17].

3. Application semiotics

Application semiotics stresses the need for semiotics originating from and deployed in applications. Its diverse use comes from its design for a family of products [3]. It is not intended as *generic semiotics* for some intelligent application to adopt or plug into, but *flexible semiotics* that allows for topological and epistemological variability and partial reusability. The rationale is well stated in the distinction between ‘generic architecture’ and ‘highly flexible architecture’ in Organization Domain Modelling [26].

The application semiotics exhibits important affinity with the structure of natural language. The semiotic potential of the natural language is agreed on and shared for communication. Yet it allows for room of individual creativity. It has stable core but is capable of meeting the need of changes and diversity. The symbolic under-specification is the mechanism that enables the natural language to serve effectively as communication system with limited means for unlimited scenarios and communication acts. Its bounded set of semiotic resources is under-specified semantically, with only generic reference, to achieve unbounded potential of specific reference in a given context. By the same token it provides reusability to enable the versatility of multi-variant conceptualization.

3.1. Ontology-based approach

Ontology is an approximate shared semiotic representation of a subject matter. To fulfill the above mentioned requirements, the DOGMA¹ approach [14, 19, 20] to ontology engineering [5] is adopted with intention to create flexible, reusable bounded semiotics for diverse computational purposes for unbounded pragmatic possibilities. It distinguishes two layers of modelling to create *lexons* and *commitments* respectively. The under-specification of lexons underpins their reusability across computational tasks, applications and perspectives. The lexon commitment guarantees the specification needed for semantic consistency and well-formedness in a particular application.

¹ DOGMA stands for **D**eveloping **O**ntology-**G**uided **M**ediation for Agents.

3.2. Lexon and Lexon Base

Lexons represent binary relationship between two entities. They are the vocabulary (not terminology) of the application semiotics. Similar to the vocabulary of the natural language, they have ideational purport without reference to specific application or task contexts. They are the potential and means of the semiotic system yet to be contextualized, deployed and fully specified meaningfully. Thus underspecified, they serve as basis for consensus, agreement, reusability and versatility.

A *lexon* is a quintuple $\langle \gamma, t_1, r_1, t_2, r_2 \rangle$, where $\gamma \in \Gamma$ is a context identifier, $t_1 \in T$ and $t_2 \in T$ are terms referring to the entities in a semantic relationship. $r_1 \in R$ and $r_2 \in R$ are roles in the semantic relationship. Γ , T and R are strings over an alphabet, Σ^+ .

```
<OrderProcessing, OrderManager, select, OrderSupplier, selected>
<OrderProcessing, AccountsManager, determine, PaymentMethod, determined>
<OrderProcessing, AccountsManager, check, CustomerStatus, checked>
<OrderProcessing, AccountsManager, send, DeliveryNote, sent>
<OrderProcessing, Customer, receive, DeliveryNote, received>
```

The context identifier, *OrderProcessing*, indicates an ideational context in which terms and roles become meaningful. The ideational context is externalized by a set of resources, such as documents, graphs, databases. Through this resource, the semantic extension of a lexon is established, communicated, documented and agreed upon among ontology developers. With specified ideational contexts, the lexons are not unspecified: they are not merely syntactic by nature. They are under-specified, representing the type rather than the token in the application domain. They do not include axioms or constraints that guarantee the semantic soundness for computation or reference to particular instances of *OrderManager* or *AccountsManager* in an application. They can be reified to represent particular viewpoints: in the above cases, action, data and organizational views of business processes.

Table 1. Reification of lexons

Action View $\langle \gamma, r_1 \rangle$	Data View $\langle \gamma, t_2, r_1 \rangle$	Organizational View $\langle \gamma, t_1, r_1 \rangle$
Select	Select_OrderSupplier	OrderManager_Select
Determine	Determine_PaymentMethod	AccountsManager_Determine
Send	Send_DeliveryNote	AccountsManager_Send
Receive	Receive_DeliveryNote	Customer_Receive
Check	Check_CustomerStatus	AccountsManager_Check

The use of lexons follows the principle of minimal encoding bias and minimal ontological commitment [7] with no assumptions of formal language representations or how the semiotics is to be structured in data structure.

Lexon base is a bag of lexons, unordered and unstructured. It is a potential in terms of which the application semantics is to be architected and feature-constrained. In analogy to natural language system, its semantic coverage is inconsistent, ambiguous, overlapping, contradictory and redundant. It embodies

multiple subject worlds as well as the multi-dimensional and multi-perspective presentation of the one and same.

3.3. Commitment Statement and Discourse

Lexons become fully specified in the pragmatic context on the *commitment layer*. The meaning of *commitment* here is application-specific interpretation of lexons. The processing agent in a given application context commits to a selected set of lexons with constraints and organized in particular networks. It depicts the application-specific tokens or instances of generic types and classes modeled in the lexon base. Here the lexons become fully specified, consistent and unambiguous, specific to particular task or application or service. The *ideational context* is semantic whereas the *application context* is pragmatic with specific references, in a given task sequence, for a particular functionality and in a given system context.

A *commitment statement* is a lexon augmented with application-specific feature constraints. It is of a tripartite structure: *theme*, *transition* and *rheme*. The theme and rheme are filled with the terms in the lexon and transition is one of the two roles. It has a narrower denotation than the subject-predicate-object statement in RDF [23]. The fillers of themes and rhemes are only resource not value as in RDF terms. The names are borrowed from functional schools of linguistics [6, 11] to emphasise the functional, pragmatic and network perspectives of lexons in particular application contexts. A lexon <ABCDE-1-3.2, SaleOffer, CharacterisedBy, Validity, Characterize> can be turned into a commitment statement as follows.

```
STATEMENT
<ABCDE-1-3.2, SaleOffer, CharacterisedBy, Validity, Characterize>
THEME Validity WITH value:true, min:1, max:m
TRANSITION Characterize
WITH aspect:progressive, durationValue:2, durationUnit:month
RHEME SaleOffer WITH min:1, max:1
END
```

A set of application-specific commitment statements is a projection or view of the lexon base. Each take a particular perspective in the role selected in the transition of the statement. The key word, *with*, introduces a list of attribute value pairs as constraints of cardinality, reference scheme, etc.

Commitment statements are connected to each other into *commitment discourse*, using operators such as those of set and logic relationship, sequence and operational procedures.

```
DISCOURSE accept_purchase_request (IN customer)
VAR request
EVENT
STATEMENT <ABCDE_1-2, Customer, Send, PurchaseRequest, BeSent>
THEME PurchaseRequest WITH ref:request
TRANSITION BeSent
RHEME Customer WITH ref:customer
END
ACTION
DISCOURSE check_customer (customer, request)
BEFORE DISCOURSE check_payment_method (customer, request)
AND DISCOURSE check_items (request)
DISCOURSE notify_client_on_purchase_request (customer, request)
END
```

The example captures the business logic: at the event of the customer sending a purchase request, the customer status must be checked before the payment method is selected. The required product items must also be checked in parallel possibly. Finally notification is sent to customer on the purchase request.

The structural backbone of commitment discourse is based on four main dimensions of network connection:

- Data flow connection between discourses in the form of input and output parameters and variables
- Connections by logical/structural operators among statements and discourses
- Discourse embedding
- Inter-statement feature constraints

4. Life cycle

In recognition of important commonalities with software development methodology, we adopt the RUP life cycle model [12, 15, 16] to phase the seven ASEP activities into inception, elaboration, construction and transition. While the documentation and validation are the activities going through all the phases, there are different degrees of focus on the problem determination, scoping, analysis, development and deployment in each phase. The darker shading indicates our observation of the intensity of work of a given phase in Figure 2.

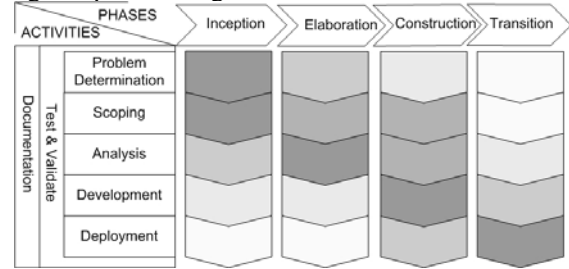


Figure 2. Lifecycle of ASEP

The inception phase studies the problem space and determines solution strategies. It seeks to scope and cut up the problem space for modeling and activity management. The elaboration phase consolidates the scope of each modelling attempt and produces the structured and detailed analysis of business logic or knowledge in the problem space. The construction phase models ontology and its application from analyses. The transition phase deploys the ontology in an application-specific form or platform. These activities are iterated with phases.

5. Activities and deliverables

The activities and deliverables of the scoping, analysis and development are designed to facilitate collaborative engineering to build consensus among stakeholders and

developers. It stresses the traceability of engineering decisions for effective communication in multi-disciplinary development teams.

5.1. Scoping with stories

Scoping the problem space for modeling application semiotics is a significant stage for effective and focused development, especially in ill-structured multi-disciplinary domains. The instrument to manage the scope of application semantics for modeling is stories.

The story is a semantically rich use case that describes a unit of knowledge or business logic, identified in knowledge elicitation. Its purpose is to communicate and document the focus of attention in the semantic domain. It is the starting point of a new or iteration of knowledge modelling task. It serves similar purpose as motivating scenarios [8] and central role as UML use cases. It consists of

- meta-data about the story authoring,
- purpose to summarize the intention of the story
- settings to specify business context and assumptions
- characters: actors and objects in the story
- episodes to specify declaratively or procedurally parts of business logic or domain knowledge
- annotation for notes

STORY			
Title	Order processing semantics for ABC Cooperative	ID	ABCDE-ST1
Theme	Ontology of business data and processes		
Author	BS Analysts, Inc		
Source	ABCDE Cooperatives		
Status	Validated	Date	13 December 2003
Purpose	BS Analysts, Inc proposes a web service solution to enable the interoperation over internet for ABCDE Cooperatives to for the foreseeable expansion, adaptability and near-partner integration to face up dynamics and competition in supply chain management.		
Settings	<p>S1 ABCDE Cooperatives is an international business partnership network of SME manufacturers, warehouses, service firms managing customer credits, accounting, and retailers.</p> <p>S2 In order to adapt to the speedy change in supply and demand relationship, meet a wide variety of customer needs, improve time-to-market, price and delivery, reduce management cost, they decide to make their respective information systems interoperate in their strategic network.</p> <p>S3 They envision that their interoperative information system network in peer-to-peer communication and coordination in task accomplishment.</p> <p>S4 The cooperative information system is required for flexibility and balance between commonality and individuality.</p> <p>S5 The cooperation information system shall be open and adaptable to allow for the expansion of business lines and activities and the participation of new partners.</p>		
Characters	<p>C1 Companies <i>As</i>, in charge of retail answers queries about the products, take order requests and manage customer accounts.</p> <p>C2 Companies <i>Bs</i> specialize in order expedition and stock management.</p> <p>C3 Companies <i>Cs</i> provide warehouses and maintain an up-to-date inventory of stock.</p> <p>C4 Companies <i>Ds</i> receive manufacture orders from Companies <i>Bs</i> and deliver to Companies <i>Cs</i>.</p> <p>C5 Companies <i>Bs</i> receive transportation requests from <i>Cs</i> to deliver stock and orders.</p> <p>C6 <i>Purchase request</i> records the customer information about an order. It consists of information on the name and address of customers, items requested, payment methods, date of request and sales venue.</p> <p>C7 <i>Requested items</i> are described in terms of codes in the catalogue, description, price and quantity, availability.</p> <p>C8 The <i>order record</i> consists of order number, customer, sales venue, delivery location, total items ordered, price.</p> <p>C9 <i>Order requisition note</i> informs on supplier, items ordered, order number, date the requisition issued, date of delivery.</p> <p>C10 <i>Sales commission</i> is calculated with information on sales venues, suppliers, total sales value, percentage.</p> <p>C11 <i>Customers</i> enquire, order, pay and receive products from ABCDE Cooperatives.</p> <p>C12 <i>Products</i> are entries offered for purchase in the catalogues of ABCDE Cooperatives.</p>		
Episodes	<p>E1.0 Customers query about products. <i>As</i> apply necessary information from product catalogues.</p> <p>E1.1 <i>As</i> check the customer credit and dispatch orders at the customers' purchase requests to <i>Bs</i>.</p> <p>E1.2 <i>Bs</i> check and make sure the availability of products either from <i>Cs</i> or <i>Ds</i>.</p> <p>E1.3 Informed of the availability status of the order, <i>As</i> manages the cancellation or negotiates partial delivery of orders.</p> <p>E1.4 When customer order confirmed, <i>Bs</i> send delivery or manufacture advice to <i>Cs</i> or <i>Ds</i>, depending the availability.</p> <p>E1.5 When ordered items are available, <i>Cs</i> informs <i>As</i> of the delivery and requests <i>Bs</i> to deliver the order.</p> <p>E1.6 When manufactured, the product is transported by <i>Bs</i> to <i>Ds</i> at the request of <i>Cs</i>.</p>		
Annotation	<p>AC12 Products are catalogued by different <i>As</i>.</p>		

Figure 3. A story of business processes

5.2. Analysis

The analysis can be likened to drawing a map of the problem space brought to focus by the story. The aim is to create a conceptual model of the part of domain

conceptualization under consideration. The strategy is to divide and conquer. The steps are decomposition and elaboration. The model decomposes the conceptualization into hierarchical structures to manage modelling complexity. A familiar example in software engineering is activity decomposition diagram [24]. Below is an example of business process breakdown.

```

1. Sales
1.1 Query products
1.2 Answer queries about products
1.3 Accept purchase request
1.3.1 Verify purchase request
1.3.2 Respond to purchase request
2. Accounting
2.1 Verify customer status
2.1.1 Check customer credit
2.1.2 Determine payment method
2.2 Receive order
2.3 Send invoice of the order
2.4 Receive payment
2.5 Update customer credit
2.6 Calculate sales commission
3 Order fulfillment
...

```

Each constituent of the conceptual model is elaborated in natural language. Unit 1.3.1 can be elaborated as

```

BEGIN Accept purchase request
IF each item is listed in catalogues
    AND IF each item is NOT suspended from catalogues,
    AND IF each item has complete and accurate specification
THEN customer purchase request is verified
ELSE customer purchase request is NOT verified
END

```

The analysis is conducted of documentation, manuals, legislature, protocols of elicitation, by business analysts, knowledge analysts and domain experts. The resultant conceptual model is ‘informal scheme’ [1], elaborated in plain and straightforward natural language, in a terminology of particular subject matter.

5.3. Development

The development of application semiotics takes the result of scoping and analysis as input to produce disciplined schemes [1]: a set of lexons and their commitments.

Developing lexons

Its main tasks are *extraction*, *abstraction* and *organization*. The extraction of lexons is text-based, taking the result from analysis as input. It spots key words and phrases in a given text in a natural language. The exercise is largely linguistic and similar to skip n’ span of the fast reading. The highlighted words in the following two examples are spotted as key words to be considered at the step of abstraction.

```

IF offerers who make a public offering did not give advance notice thereof to stock exchange regulator, attaching the prospectus to be published
THEN The offerers are conducting unauthorized solicitation of investors

```

The highlighted text, as a working document, provides two important services here. One is that it provides a tangible scope of work at a particular time of development. The other is that it becomes a record of decision-making, traceable and visible across time, location and teams.

The abstraction is a process of postulating abstract conceptions of terms, roles and lexons. The extracted

words and phrases are linguistic embodiment of concepts to be modelled. The surrounding text conveys the context for understanding and communication of concepts. Since it is based on the highlighted text, there is a clear borderline imposed on conceptual modelling with an explicit focus of attention.

Table 2. Examples of lexons

Context	Term	Role	Role	Term
D.58.94.1	Offerer	Make	MadeBy	PublicOffering
D.58.94.1	PublicOffering	SubtypeOf	SupertypeOf	Offering
D.58.94.1	Offerer	Send	SentBy	AdvanceNotice
D.58.94.1	AdvanceNotice	Concern		PublicOffering
D.58.94.1	Regulator	Receive	ReceivedBy	Notice
D.58.94.1	AdvanceNotice	Include	IncludedBy	Prospectus
D.58.94.1	Regulator	Publish	PublishedBy	Prospectus

While the abstraction is confined to the text and a bottom up approach to modelling, the organization is a step that goes beyond the current working document, covering multiple ideational contexts. It is devoted to two main purposes. One is to structure the lexons extracted and abstracted bottom up in the previous steps, such as merging or introducing subtyping relationship. The other is to integrate the lexons into existing semiotic systems, such as upper or foundational ontologies.

Developing commitments

The development of lexons uncovers abstract conceptual types from the story and analysis model. Having established conceptual types, the development of commitments is to come back to the ground, modelling their tokens. While lexons underpins the flexibility and reusability of the application semiotics with under-specification, the commitment is essentially dedicated to the semantically well-formed, fully specified, consistent actualizations of the underlying patterns with respect to a particular task or application. The fully specified semantics in the commitments depends on their pragmatics: tasks and application context. The activity takes a different point of view of the results of scoping and analysis. It seeks to capture specific business or knowledge entities, processes or patterns in terms of lexons, so that specifics can be interpreted, marshalled, organized or interoperated in term of generics.

The development of commitments involves four steps: *select*, *focus*, *constrain* and *connect*. It first delineates the semantic space by selecting a set of lexons from the lexon base. Each lexon is tokenized into commitment statement with a focus. A lexon, $\langle \gamma, t_1, r_1, t_2, r_2 \rangle$, can be focused in the form of $[t_1, r_1, t_2]$ or $[t_2, r_2, t_1]$, depending on the choice of the role. The terms and roles are then constrained to refine or confine the semantic references and properties as required in an application context. The commitment statements are finally connected into a network with set, logical and operational operators. Below is an example of the commitments representing data objects in business processes.

```
DISCOURSE purchase_request
STATEMENT <ex2, PurchaseRequest, CharacterisedBy, CustomerName, Characterize>
THEME CustomerName WITH min:0, max:m
```

```
TRANSITION Characterize
RHEME PurchaseRequest WITH min:1, max:1
END
AND
STATEMENT <ex2, PurchaseRequest, CharacterisedBy, CustomerAddress, Characterize>
THEME CustomerAdress WITH min:0, max:m
TRANSITION Characterize
RHEME PurchaseRequest WITH min:1, max:1
END
AND
STATEMENT <ex2, PurchaseRequest, CharacterisedBy, PaymentMethod, Characterize>
THEME PaymentMethod WITH min:1, max:m
TRANSITION Characterize
RHEME PurchaseRequest WITH min:1, max:1
END
AND
STATEMENT <ex2, PurchaseRequest, CharacterisedBy, PurchaseItem, Characterize>
THEME PurchaseItem WITH min:0, max:m
TRANSITION Characterize
RHEME PurchaseRequest WITH min:1, max:m
END
AND
STATEMENT <ex2, PurchaseRequest, CharacterisedBy, ReceptionDate, Characterize>
THEME ReceptionDate WITH min:1, max:m
TRANSITION Characterize
RHEME PurchaseRequest WITH min:1, max:1
END
END
```

The layered model of application semiotics is important for encapsulating changes and dynamics of models. For example, the continued business process improvement or integration can be catered to by optimisation (different commitment constraints), restructuring (changes in commitment networks), or innovation (new business concepts with additions of lexons).

5.4. Deployment

The development of commitments models in the perspectives of application purposes and tasks. It does not take into account another pragmatic aspect: system context or computational platforms. The deployment stage considers where the application semiotics is to be used and the format needed to deploy the lexons and commitments. The commitments are treated as the specification of application semantics to be handed over to the application system engineer to load in the application systems.

For example, the application semiotics of business processes can be transformed into BPML4WS, BPML. The commitments of web service description can be translated into OWL or DAML-S.

6. Conclusion

ASEP recognises the need a different track of development for engineering application semantics. The complexity of application semantics requires it to be handled in its own iterative development cycle and management, rather as part of conventional software development based on the paradigm of ‘close ontology’ [1] before the main loop or scattered in the main iteration of software development [16]. It is envisaged as parallel track to system engineering track as in DE [27].

Compared with classical knowledge engineering approaches [9, 18, 25], ASEP emphasises domain

ontology modelling, instead of going straight to code knowledge rules from the result of analysis. This extra effort is justifiable with aims of reusability and maintainability of business logic in dynamic multi-dimensional application domains. It is necessary for a development involving distributed multi-discipline teams intending to cover product-lines of systems. It is desirable for applications requiring complex application semiotics such as large scale knowledge systems, where the effective management and visualization of the existing rules is crucial for controlling modelling complexity. In order to put the knowledge/ontology engineering on the basis of disciplined team work, the traceability of modelling decision is stressed from stories through analysis models to lexons and commitments and their deployment. One may dispute its necessity and overhead on the performance of a given developer at particular moments of development, but the over-all benefits for the whole team of collaborative participants and evolution of development in the full life cycle of the project are significant and far reaching. On the other hand, ASEP is not intended as a methodology of 'high ceremony' [16] in order to make sure of agility development necessary in distributed multidisciplinary environment of development.

ASEP is intended to bind the development as late as possible to a formal language of knowledge representation and computational paradigms of particular computation semantics such as inferential, denotational or operative semantics. It considers these issues in conjunction with the pragmatics of application tasks and deployments.

Acknowledgement

This study is partially funded by the EU 5th framework program, IST 2001-38248.

References

- [1] E. Compatangelo and G. Rumolo, "An engineering framework for domain knowledge modelling", *Information Modelling and Knowledge Bases IX*, IOS Press, 1989, pp. 51 -56.
- [2] J. Culler, *The Pursuit of Signs: Semiotics, Literature, Deconstruction*, New York, Cornell University Press, 1981
- [3] K. Czarnecki, *Domain Engineering*, in K. Czarnecki and U. Eisenecker, ed., *Generative Programming: Methods, Techniques and Applications*, Addison-Wesley, 1999.
- [4] F. De Saussure, *Course in General Linguistics*, New York, McGraw-Hill, 1966.
- [5] A. Farquhar, R. Fikes, W. Pratt and J. Rice, *Collaborative Ontology Construction for Information Integration*, KSL, Stanford University, 1995.
- [6] J. Firbas, *On some basic issues of the theory of functional sentence perspective*, *Brno Studies in English* 15, 1983 pp. 9 – 36.
- [7] T. Gruber, *A translation approach to portable ontology specifications*, *Knowledge Acquisition*, vol 5, no 2, 1993, pp. 199 – 220.
- [8] M. Grüninger, M. S. Fox, *Methodology for the design and evaluation of ontologies*, *IJCAI-95 Workshop on basic ontological issues in knowledge sharing*, Montreal, 1995.
- [9] G. Guida and C. Tasso, Eds., *Topics in Expert System Design - Methodologies and Tools*, North-Holland, Amsterdam, NL, 1989
- [10] R. Gudwin and F. Gomide, "Computational Semiotics : An Approach for the Study of Intelligent Systems - Part I: Foundations", *Technical Report, RT-DCA 09*, 1997.
- [11] M. A. K. Halliday, *An introduction to functional grammar* London, Arnold, 1994.
- [12] I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Boston, Addison-Wesley, 1999.
- [13] M. J. Jarke et al , *DAIDA: an environment for evolving information systems*, *ACM Transactions on information systems* Vol 10, No. 1, 1992, pp. 1-50
- [14] M. Jarrar, J. Demey, R. Meersman, "On using conceptual data, modeling for ontology engineering", *Journal of data semantics*, Vol 1, No. 1, 2003.
- [15] P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy: a Practitioner's Guide to the RUP*, Boston, Addison-Wesley, 2003.
- [16] P. Kruchten, *The Rational Unified Process: an introduction*, Boston, Addison Wesley, 2000.
- [17] S. A. McIlraith and D. L. Martin, *Bringing Semantics to Web Services*, *IEEE Intelligent Systems*, January/February, 2003, pp. 90-93.
- [18] M. McTear and T. Anderson, *Understanding Knowledge Engineering*, Chichester, Ellis Horwood, 1990.
- [19] R. Meersman, *Semantic Ontology Tools in IS Design*, *ISMIS'99*, Warsaw, 1990.
- [20] R. Meersman, *Reusing certain database design principles, methods and techniques for ontology theory, construction and methodology*, *STARLab VUB Technical Report 01*, 2000.
- [21] C. Morris, *Foundations of the Theory of Signs*, Chicago, University of Chicago Press, 1938.
- [22] C. S. Peirce, "Logic as Semiotic: The Theory of Signs", J.Bucher, ed., *Philosophical Writings of Peirce*. New York:,Dover, 1955.
- [23] *RDF Resource Description Framework*. <http://www.w3c.org/RDF/> .
- [24] Rock-Evans, *Data modelling & process modeling*, Oxford Butterworth-Heinemann, 1992.
- [25] G. Schreiber, et al, *Knowledge engineering and management: the CommonKADS Methodology*, London, MIT, 2000.
- [26] M. Simos, D. Creps, C. Klinger, L. Levine and Allemang, *Organization Domain Modeling Guidebook*, Version 2.0 STARS-VC-A025/001/00, 1996.
- [27] *Software Engineering Institute, Model-based Software Engineering*, <http://www.sei.cmu.edu/technology/mbse>, 1997.
- [28] P. Spyns, R. Meersma, *From knowledge to interaction: from the Semantic to the Pragmatic Web*. STARLab, VUB Technical Report.

Applying Aspect-Orientation in Designing Security Systems: A Case Study

Shu Gao, Yi Deng, Huiqun Yu, Xudong He, Konstantin Beznosovⁱ, Kendra Cooperⁱⁱ

School of Computer Science, Florida International University

ⁱDepartment of Electrical and Computer Engineering, University of British Columbia

ⁱⁱDepartment of Computer Science, University of Texas at Dallas

{sgao01, deng, yhq, hex}@cs.fiu.edu; ⁱbeznosov@ece.ubc.ca; ⁱⁱkcooper@utdallas.edu

Abstract. *As a security policy model evolves, the design of security systems using that model could become increasingly complicated. It is necessary to come up with an approach to guide the development, reuse and evolution of the design. In this paper, we propose an aspect-oriented design approach to designing flexible and extensible security systems. A case study demonstrates that such an approach has multifold benefits and is worth further exploration.*

1. Introduction

A security policy model always evolves; accordingly, the design of a security system using that policy model should reflect the changes. Using role-based access control (RBAC) as an example, currently it supports role hierarchy, static separation of duty relations, and dynamic separation of duty relations. As research on RBAC progresses, more concerns have been and will be covered. So the model hierarchy of RBAC is quickly becoming more and more complicated, which requires that the security system supporting RBAC be flexible and extensible. To address this issue at the design level, we propose an aspect-oriented approach to designing flexible and extensible security systems. This paper illustrates the approach through a case study, which is part of a design for CORBA access control (AC) supporting RBAC models.

Although some papers in the literature have dealt with separating security concerns in application system design, little research has been done to explore the use of aspect-orientation in designing security systems. Our work is a first step toward a systematic aspect-oriented approach to advance the design of security systems.

2. A Case Study

The CORBA AC [13] is a reference model for enforcing access control in the middleware layer of distributed applications. It is aimed to provide a standard way to

separate access control and application logic. CORBA AC specification is policy neutral in that only essential and general access control interfaces are specified. To implement a functional CORBA AC mechanism, certain access control policy models have to be supported. In this case study, we choose RBAC models, which have been widely recognized as a well-defined general approach for access control in large-scale authorization management.

2.1. Problem Analysis

In [14], the RBAC96 family contains four models: RBAC₀, RBAC₁, RBAC₂, and RBAC₃. RBAC₀ is the base model that contains (1) entities – users (U), roles (R), permissions (P); (2) static relationships – user assignment (UA – between users and roles), permission assignment (PA – a between roles and permissions); and (3) dynamic relationship – sessions (S) (a one to many relationship between a single user and his/her multiple roles). RBAC₁ extends RBAC₀ with a hierarchical structure representing the partial order relation on roles. RBAC₂ extends RBAC₀ with constraints on entities such as conflicting roles as well as relationships such as a user can only assume a limited number of roles. RBAC₃ is the combination of both extensions of hierarchy and constraints such that constraints can be defined on roles at the different levels of the hierarchy.

Since RBAC₁ to RBAC₃ are derived from RBAC₀, one design issue is how to effectively reuse the design for RBAC₀ to realize RBAC₁ to RBAC₃. The RBAC family is still evolving. The number of RBAC models is increasing to cover a variety of emerging concerns and specific application needs. For example, in the proposed RBAC standard by NIST [4], the time concern is incorporated into the concept of dynamic separation of duty relations (DSD), while the old constraint model was called static separation of duty relations (SSD). Very likely, context concern will also be introduced in the near future. If we follow the conventions used in [14], we can illustrate the evolution of RBAC family with Figure 1.

a. The RBAC96 Model Hierarchy

b. The RBAC Model Hierarchy with Time and Context Concerns

2.2. Design Approach

Recently, a new software implementation paradigm called aspect-oriented programming (AOP) based on the principle of separation of concerns was proposed [7], which has generated extensive research interest. As Kiczales et al. point out in [7], existing programming languages including procedural, functional, and object-oriented languages decompose a system into functional components. However the implementations of some properties (e.g. synchronization, real-time constraints, error handling, audit, security enforcement) cannot be encapsulated into a single component. Frequently classified as “crosscutting properties”, these properties are usually present in more than one functional component. Implementations of such properties in mainstream languages necessarily result in tangled code. Code tangling denotes the use of a single method to implement multiple properties. The purpose of AOP is to provide mechanisms that explicitly capture crosscutting structures, so crosscutting concerns can be encapsulated.

- (1) The identification of aspects;
- (2) The notations used to specify aspects;
- (3) The rules to compose aspects together.

For this case study, we regard each concern in RBAC models as an aspect and thus we have four aspects: role hierarchy (RH), static constraints (SSD), temporal constraints (DSD), and spatial constraints (SC). These four aspects are orthogonal and are faithful reflections of the separation of concerns principle. With this aspect-oriented view, the development of RBAC models will be incremental and compositional. For example, RBAC_{13} (Figure 1.b) will be built by integrating the base model RBAC_0 with aspects RH, DSD, and SC. Therefore this approach will greatly enhance the reusability of the base model and aspects, as well as provide great flexibility for RBAC evolution to meet new system needs. Thus we have a nice and elegant solution to issue (1).

2.3. AspectJ and UML Extension

Join point: A predictable point in the execution of an application.

¹ The RBAC₃ in RBAC96 family is now RBAC₅ in the extended RBAC family.

Pointcut: A structure designed to identify and select join points within a program.

Advice: Code to be executed when a join point is reached in the application code.

Inter-type declaration: A powerful mechanism to add attributes and methods to previously established classes.

Aspect: A structure analogous to an object-oriented class that encapsulates join points, pointcuts, advices, and inter-type declarations.

Join point, pointcut, and advice are used to realize dynamic crosscutting. The join point is a well-defined point in a program where another concern will crosscut this program. It can be method calls, constructor calls, method call execution, constructor call execution, field get, field set, exception handler execution and other points in the execution of a program. AspectJ uses a designator that takes a join point as a parameter to tell the aspect-oriented program when it should match the join point. The pointcut is a structure to group such designators. Whenever a join point is matched by a designator, the pointcut containing it is triggered. Some advice defined for the triggered pointcut will be executed. Depending on the type of the advice (before, after or around), the code in the advice is executed before, after, or in place of the join point. Inter-type declaration is for static crosscutting. New attributes and methods can be added to existing classes without having to explicitly modify the classes. AOP introduces a new component type – aspect. The aspect is used to encapsulate crosscutting concerns. It contains the join points, pointcuts, and advices.

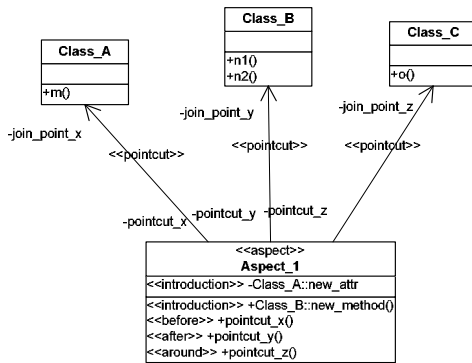


Figure 2. Extension of UML class diagram

We informally extend UML notations to model aspect-oriented design (Figure 2)². An aspect is a regular class with the newly created stereotype `<<aspect>>`. An inter-type declaration has a new stereotype `<<introduction>>`. It is like an attribute or a method in a regular class, except that its name should start with the name of the target class/aspect to which the new attribute/method is introduced. Advices have the stereotypes of `<<before>>`,

`<<after>>` and `<<around>>`. An advice has no name. The name after `<<before>>`, `<<after>>` or `<<around>>` is the name of the pointcut for which an advice is defined. A pointcut is represented by one or more navigated association(s) from an aspect to a class/aspect which the aspect crosscuts. The pointcut's name is labelled at the crosscutting aspect side. The join point's name is labelled at the side of the class/aspect being crosscut.

2.4. The Aspect-Oriented Design

Based on the above discussion, this subsection introduces an aspect-oriented design for CORBA AC that operates with $RBAC_{0-3}$ in the $RBAC_{96}$ family. It is not our purpose to present a complete and detailed design here; instead, we would focus on demonstrating how AOD realizes the separation of concerns principle, and how it helps to manage the complexity shown in Figure 1.

Base Design – Main Concern

As we have analyzed in subsection 2.1 and 2.2, the main concern of this case study is to realize a CORBA AC mechanism that supports $RBAC_0$. The design of the main concern will be reused and crosscut by the design of new concerns, therefore it is called the base design. When working on a design, it is better to have some knowledge of other concerns that may arise. However, it is always the case that the designers hardly know what will happen in the future. The good news is that, with AOD, we do not have to worry about other concerns.

Aspect One – Role Hierarchy

Let us see what new attributes and methods need to be introduced and which existing methods need to be modified to support role hierarchy. First, as a direct result of role hierarchy, functions used to manage the partial order relation are needed: `add_inheritance()`, `delete_inheritance()`. They should be added to the Role class in the base design. Consequently, the Role class needs to maintain a list of immediate ascendants and a list of immediate descendants. Second, in the base design, there is a method `get_assigned_roles(user)` in the UAList class, which returns all roles assigned to the given user and is used to determine a user's access permission to resources. When role hierarchy exists, `get_assigned_roles(user)` cannot return all roles that a user actually has, since some roles not assigned can be inherited. For example, in a bank, the role manager inherits the role employee. If John is assigned to be the manager, then he is also a bank employee though he is not explicitly assigned to that role. The access control system needs to find all roles a user actually has in order to determine the user's permissions correctly. Therefore, we add `get_authorized_roles(user)` to the UAList class for returning all roles including the inherited ones of a user.

² Some ideas are borrowed from [15].

Similarly, we need `authorized_users(role)` (in the `Ualist`) and `authorized_roles(user)` (in the `UA` class) to take the place of corresponding “assigned_” ones in the base design. Accordingly, in the base design, two methods that used to call `get_assigned_roles(user)`: `authenticate()` from the `PrincipalAuthenticator` class and `set_roles()` from the `Credentials` class, now have to be modified to call `get_authorized_roles(user)`.

The concern to support role hierarchy crosscuts the main concern in that it cannot be implemented in a localized way with vanilla object-oriented approach (Figure 3). Several classes in the base design need to be modified or extended. On one hand, the crosscutting problem makes it expensive to modify; on the other hand, the resulting design is hard to understand and maintain.

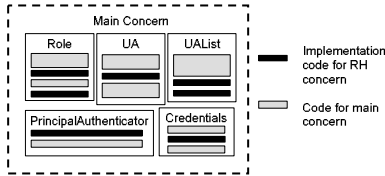


Figure 3. Tangled implementation of RH concern

With AOD, we can address this problem by explicitly representing crosscutting, and encapsulate the crosscutting concerns into aspects. The AOD class diagram for implementing $RBAC_1$ is shown in Figure 4. In the figure, two dashed frames are used to indicate the design for the main concern and the design for the role hierarchy concern respectively. Since the base design is too large, only those classes directly affected by adding the new concern are listed here and relationships other than crosscutting are omitted. As it shows, the

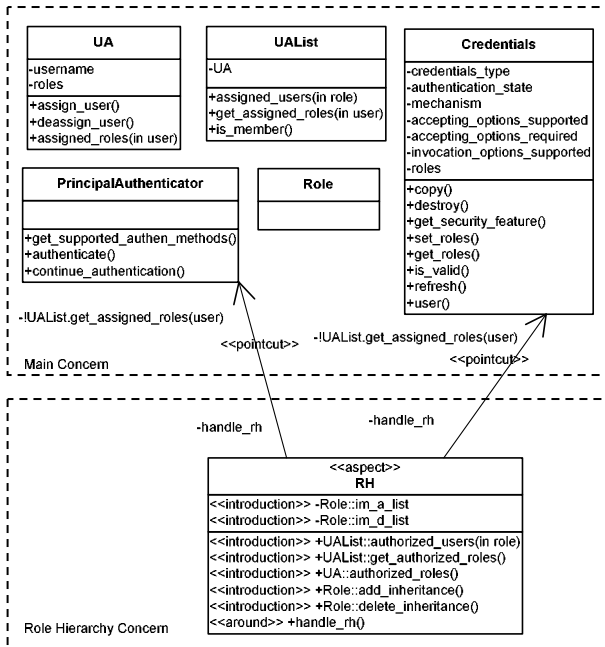


Figure 4. AOD for implementing $RBAC_1$

implementations of two concerns are well modularized without any tangling. An aspect called `RH` contains all the implementation of the `RH` concern. Inside the `RH` aspect, several inter-type declarations are defined to insert new attributes and methods into existing classes. Only one pointcut `handle_rh` and one join point `!Ualist.get_assigned_roles(user)` (“!” means it is a method call type join point) are defined. At runtime, any method call to `Ualist.get_assigned_roles(user)` generated by `PrincipalAuthenticator` or `Credentials` instance will trigger the `handle_rh` pointcut. The `<<around>>` type advice code defined for `handle_rh` will then be executed in place of the `Ualist.get_assigned_roles(user)` method. In this design, the advice code will call `Ualist.get_authorized_roles(user)` which is defined in the same aspect.

Aspect Two – Static Constraints

$RBAC_2$ allows security administrator to set static separation of duty constraints on the assignment of users to roles. In [4], an SSD constraint is defined in the form of (rs, n) where rs is a role set, and n is called “cardinality” which is a natural number ≥ 2 . (rs, n) means that no user is assigned to n or more roles from the set rs .

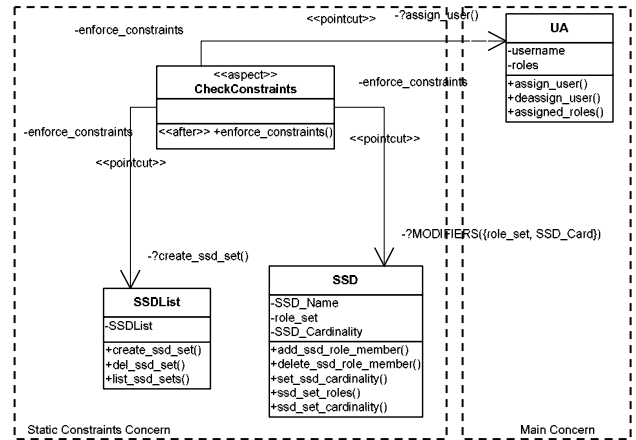


Figure 5. AOD for implementing $RBAC_2$

To implement $RBAC_2$, first we need several functions to manage SSD constraints. They are: `create_ssd_set()`, `add_ssd_role_member()`, `del_ssd_role_member()`, `del_ssd_set()`, `set_ssd_cardinality()`, `list_ssd_sets()`, `ssd_set_roles()`, and `ssd_set_cardinality()`. Besides these, every time the SSD relation or the user-role assignment relation is modified, the system must check whether the SSD constraints have been broken. So there should be a function to enforce these constraints.

It is worth noticing that the management functions for SSD constraints do not crosscut the base design. They are newly defined functions and do not need to be inserted into any classes in the base design. Should they be encapsulated into an aspect structure? We prefer not,

since we can define two new classes: SSD and SSDList, which can encapsulate these functions quite well.

The implementation of RBAC₂ crosscuts the main concern only at the point where assign_user() of the UA class is executed. A method call to the function that enforces SSD constraints need to be added after the execution of assign_user().

The function enforcing SSD constraints crosscuts SSD and SSDList class, because these two classes contain methods that may change the SSD relation.

Thus, we design an aspect CheckConstraints. In this aspect, there is a pointcut enforce_constraints. An <<after>> type advice is defined for this pointcut. Inside the advice is the code enforcing SSD constraints. There are several join points defined. All of them are of method call execution type (which will be represented by “?” in the diagram). Specifically, the execution of SSDList.create_ssd_set(), UA.assign_user(), and any methods in SSD class that modifies the role_set or SSD_Cardinality attribute will trigger the enforce_constraints pointcut.

The aspect-oriented design for static constraints concern is shown in Figure 5. Although the static constraints concern is not implemented by one aspect, but by two classes and an aspect, the implementation of this concern is still well modularized.

Composition Design – RBAC₃

RBAC₃ combines role hierarchy and static constraints. Now the advantage of AOD is obvious. By composing the base design, Aspect One and Aspect Two together, with

minor modification and without destroying current modularity, we get the design for RBAC₃ (Figure 6). According to the composition rule of AspectJ, the aspect RH dynamically crosscuts the aspect CheckConstraints. This is because the advice code enforcing SSD constraints used to call get_assigned_roles(user) to find a user's roles. With the existence of role hierarchy, now get_assigned_roles(user) should be replaced by get_authorized_roles(user). We also need to define a new join point, which is the execution of Role.add_inheritance(). It will trigger the enforce_constraints pointcut. In the figure, two <<pointcut>> associations from RH to CheckConstraints and from CheckConstraints to Role reflect these modifications.

3. Related Work

Aspect-oriented programming is an emerging technology. Recently the research on how to extend this paradigm to design level has attracted more and more attention [3, 16, 17]. The application of AOD to security domain is promising. However, research results are rare. Both [2] and [9] point out that the separation of concerns principle can be used to separate security concerns from application concerns. This is an important and relatively obvious application of AOD to security. Unlike them, we explore the use of aspect-orientation to advance the design of security systems. Due to the novelty of AOD, virtually no research has been done in this direction.

A number of UML extensions have been proposed to support AOD. Examples of such extensions are [11, 12,

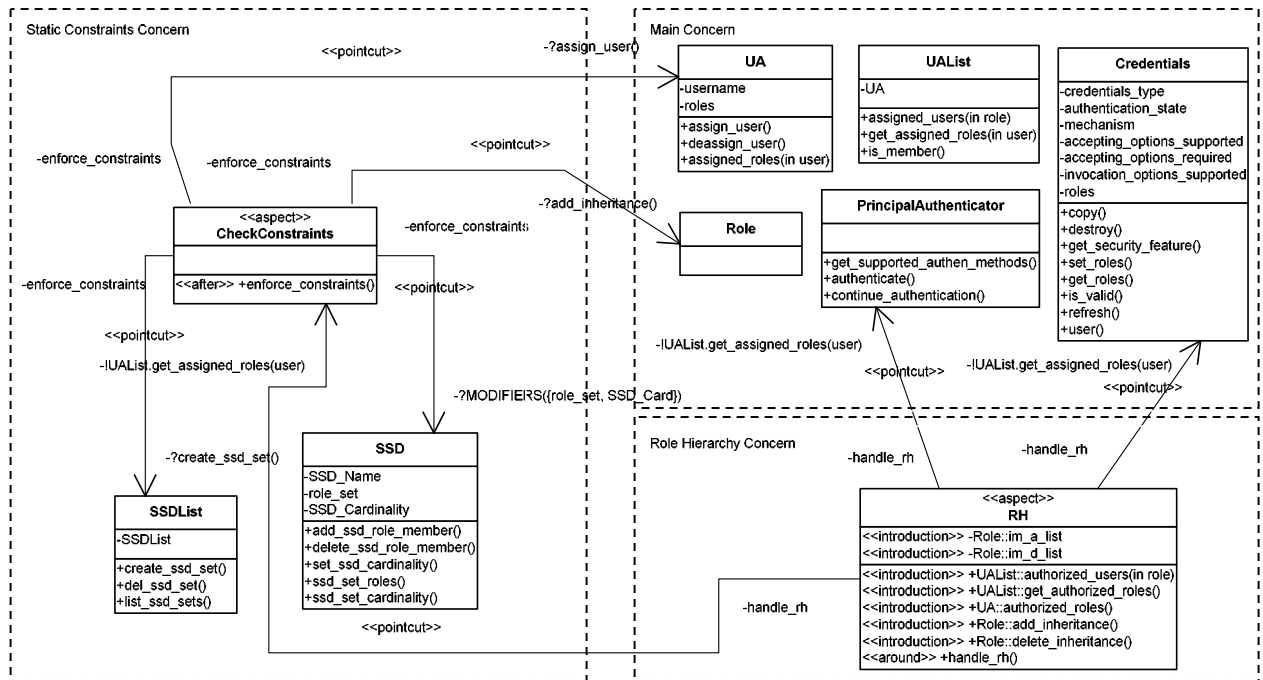


Figure 6. AOD for implementing RBAC₃

15]. So far, no extension has been widely accepted. This to some extent hampers the application of AOD. Based on the belief that UML notation should be easy to read and understand, we introduced some stereotypes with [15] as an aid for describing the CORBA AC design.

There is little work reported on implementing RBAC in CORBA systems. The design in this paper is based on our previous research, described in [10], which shows that CORBA Security architecture is capable of supporting RBAC₀ – RBAC₃ and determines strategies for implementation. However, it does not propose a specific design of CORBA Security. Using one of the strategies from [10], this paper suggests a specific way for implementing RBAC96 model on CORBA systems.

4. Conclusion

The principle behind AOD is separation of concerns. By applying AOD approach in CORBA AC design, a number of benefits of separation of concerns are acquired. Since RBAC extensions covering different concerns can be encapsulated using aspects, we get better modularity with the CORBA AC design. Better modularity leads to better comprehensibility, reusability, flexibility and maintainability. Because there are well defined mechanisms explicitly supporting both dynamic and static crosscutting, the design can be incrementally extended to cover temporal, spatial or other future concerns in RBAC models.

Through this case study, we propose an aspect-oriented design approach to designing security systems. Our work is a first step toward a systematic aspect-oriented approach to advance the design of security systems. Our approach is easy to learn and apply. Although we have used the composition rules of AspectJ and an extended UML design notation for the design presented, our approach does not depend on a specific implementation model.

Our next step is to apply formal methods in AOD. Formal analysis is very useful for detecting possible errors early in the design phase, which is especially important to the design of security systems.

5. Acknowledgements

This work is supported in part by NSF under grant No. CCR-0226763 and No. HRD-0317692.

References

[1] AspectJ homepage. <http://eclipse.org/aspectj/>

- [2] B.D. Win, F. Piessens, W. Joosen and T. Verhanneman. On the Importance of the Separation-of-Concerns Principle in Secure Software Engineering. In Workshop on the Application of Engineering Principles to System Security Design, December 22, 2002.
- [3] B. Tekinerdogan and M. Aksit. Deriving Design Aspects from Canonical Models. In Object-Oriented Technology, S. Demeyer and J. Bosch (Eds.), LNCS 1543, ECOOP'98 Workshop Reader, Springer Verlag, pp. 410-413, July 1998.
- [4] D.F. Ferraiolo, R. Sandhu, S. Gavrilu, D.R. Kuhn and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and System Security, vol. 4, pp. 224-274, 2001.
- [5] E.W. Dijkstra. A Discipline of Programming. Englewood Cliffs, NJ: Prentice Hall, 1976.
- [6] G. Booch, J. Rumbaugh and I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley Longman, Inc, 1999.
- [7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C.V. Lopes, J.-M. Loingtier and J. Irwin. Aspect-Oriented Programming. In Proceedings of ECOOP'97 - Object-Oriented Programming, 11th European Conference, Jyväskylä, Finland, 1997.
- [8] J.D. Gradecki and N. Lesiecki. Mastering AspectJ: Aspect-Oriented Programming in Java. Wiley Publishing, Inc, 2003.
- [9] J. Viega, J.T. Bloch and P. Chandra. Applying Aspect-Oriented Programming to Security. Cutter IT Journal, vol. 14, no. 2, pp. 31-39, 2001.
- [10] K. Beznosov and Y. Deng. A Framework for Implementing Role-Based Access Control Using CORBA Security Service. In the Fourth ACM Workshop on Role-Based Access Control, Fairfax, Virginia, USA, October, 1999.
- [11] M. Basch and A. Sanchez. Incorporating Aspects into the UML. In Proceedings of Third International Workshop on Aspect-Oriented Modeling, March 2003.
- [12] O. Aldawud, T. Elrad and A. Bader. UML Profile for Aspect-Oriented Software Development. In Proceedings of Third International Workshop on Aspect-Oriented Modeling, March 2003.
- [13] OMG. CORBA Security Service Specification, Version 1.8, March 2002.
- [14] R. Sandhu, E. Coyne, H. Feinstein and C. Youman. Role-Based Access Control Models. IEEE Computer, 29(2):38-47, February 1996.
- [15] R. Pawlak, L. Duchien, G. Florin, F. Legond-Aubry, L. Seinturier and L. Martelli. A UML Notation for Aspect-Oriented Software Design. In Aspect-Oriented Modeling with UML Workshop at AOSD 2002, Enschede, the Netherlands, 2002.
- [16] S. Clarke and R. J. Walker. Composition Patterns: An Approach to Designing Reusable Aspects. In Proceedings of the 23rd International Conference on Software Engineering (ICSE), Toronto, Canada, May 2001.
- [17] S. M. Sutton Jr. and P. Tarr. Aspect-Oriented Design Needs Concern Modeling. In Aspect Oriented Design 2002 Workshop, April 23, Enschede, The Netherlands.

Applying Ontologies in the KDD Pre-Processing Phase

Guillermo Nudelman Hess, Cirano Iochpe

Universidade Federal do Rio Grande do Sul – Instituto de Informática

{hess, ciochpe}@inf.ufrgs.br

Abstract. This article proposes a methodology for using the power of ontologies in the Knowledge Discovery in Databases (KDD) pre-processing phase. The goal is to prepare geographic database's (GDB) conceptual schemas to be mined, in order to obtain analysis patterns candidates. The ontology is applied in the schema's semantic unification, which is very important in this process, since the data mining tools are not capable to handle semantic conflicts. A methodology to refer and update the knowledge basis was developed, based on some similarity matching measurement between concepts.

1. Introduction

Because of the increasing use of Geographic Information Systems (GIS) in the last past years, the conceptual modeling of the Geographic Database (GDB) has become a very important task. Basically, a GDB differs from a traditional database by its capability to store not only conventional (descriptive) data, but also spatial, geo-referenced data.

However, each one of the GIS software has its own data model, focused basically in the logical phase of the database project [22]. Thus, the development of the GDB gets burdened to the software architecture of the GIS is going to be used.

The use of conceptual modeling allows not only the independence from the software implementation, but also the reuse of the model, or at least of part of it, several times. This reuse is specially interesting in GDB since its modeling is quite complex and part of the geographic concepts of the real world being designed is repeated for distinct applications. In this way, the use of analysis patterns [8] is useful. Analysis patterns are the essence of the conceptual modeling for the solution of a recurrent problem in a specific context.

To support the acknowledgment of analysis patterns automatically, the Knowledge Discovery in Databases (KDD) [7] may be applied. This process has several steps, as shown in Figure 1.

The main phase is the data mining (DM). However, to achieve it successfully, a preparation of the input data must be performed before. To make possible the mining

of several conceptual schemas of GDB, from different organizations and with distinct objectives, they must be integrated to solve conflicts and incompatibilities among them.

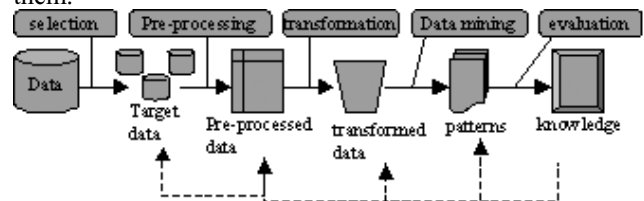


Figure 1- The KDD process [7]

To reach a correct data preparation for mining, this schemas integration must handle semantic heterogeneities which comprises the problem of unification between the concepts used to describe the real world phenomena, and the relationships among them. In this sense, it is necessary to build a Knowledge Organization System (KOS) [11], such as an ontology [18] to store the concepts concerning of the geographic applications domain. Furthermore algorithms of similarity matching have to be used to achieve correct interpretation for the variations of known terms, and classification of new ones.

It is important to clarify that in this paper we are not proposing a geographic ontology. We are adapting this concept to another to be used exclusively in the GDB environment project.

The remaining of this paper is organized as follows. Section 2 presents the context of the semantic integration problem of GDB conceptual schemas. Section 3 details how the ontology can be used in the architecture. The methodology of the semantic integration is shown in section 4. At last, the conclusions and future work are shown in section 5.

2. GDB Schema's Semantic Integration

The semantic level of heterogeneity include subjects related to the comprehension and use of data related to different applications and users, involving distinct data models and distinct interpretations of these different data models. The explanation to this fact is quite simple. The same real world entity, modeled by two or more people,

probably will not have the same modeling, even though it is representing the same phenomenon of the application's domain. In these cases occur what is called a conflict. A conflict is nothing else than a difference in the representation of the same concept.

2.1. Levels of Heterogeneity

According to Partridge [17], the semantic heterogeneity can be classified in disagreement between communities and disagreement in form.

Disagreement between communities occurs when two or more communities do not agree about the meaning of data, or of part of it, in a database. As result the communities use different words to express the same concepts of the real world. These different terms have the same meaning, and generate what is called synonym.

Disagreement in form happens when the same dataset in different databases has semantic differences. The same portion of data has distinct meaning in two or more databases. This generates what is known as homonym.

Bergamaschi et. al [3] go further in this definition, classifying the heterogeneities in two types, naming and structural. The first case comprises both aspects presented in [17], and the structural heterogeneity cover the existing differences in the conceptual model used to describe the concepts, in terms of attributes and relationships.

According to Park [16], the semantic heterogeneity can be classified, broadly, in two different levels: the schema level and the data level. In the schema level the heterogeneity is a consequence of the differences on the logical structures and/or inconsistencies in the metadata from the same domain, used in distinct databases. This is caused by the different structures (tables as attributes) used to represent the same information, and by the use of different specifications to the same structure. The heterogeneity in the schema level can be divided in six types of conflicts: naming conflicts (homonyms and synonyms), entity identification, schema isomorphism, generalization, aggregation and schematic dissimilarities.

The data heterogeneity result on the data domain differences, caused by the multiple representations and interpretations about the semantic of a data. This heterogeneity can also be divided in six categories: value, representation, unit, precision (including the granularity and spatial resolution), trust on the known data values and spatial domains conflicts.

Visser et al. [23] focus the heterogeneity problem classifying it in four distinct categories. The paradigm heterogeneity happens when two systems express their knowledge using different modeling paradigms, as, for example, one object oriented and the other based on the entity relationship model. The language heterogeneity exists if two systems express their knowledge in different languages. The ontological heterogeneity occurs when

two systems disagree over the meaning and structure of the existing elements in their application's domain. At last, the content heterogeneity happens if two systems represent totally distinct contents. The last two categories together compose the semantic heterogeneity.

Specifically in the geographic database modeling, this problems get more evident, because of the natural complexity of the geographic data [16]. GDB's target is the modeling of the reality phenomena, that is, the real world existing concepts. Hence, the set of elements to be modeled is quite restrict (small) and very concrete. The attributes and associations between the geographical elements are always the same. What changes is the approach used, which depends on the application's aim and on the designer's knowledge, and also the names used to represent the same things.

2.2. Requisites for the Integration of GDB conceptual schemas

To make the integration of geographic conceptual schemas possible, three requisites must be satisfied [3]:

- The conceptual schemas from each one of the sources has to be available;
- There should be semantic information in the schemas;
- A canonical data model has to exists. This standard model has to have enough expressiveness power to describe all the models to be integrated;

Once the target of the integration proposed in this paper is of conceptual schemas, the first requisite is automatically satisfied. The other requisites are satisfied by the use of the work developed in [1][10] and by the use of a standard format for geographic data, the GML [15].

3. The Role of the Ontology

The role of the ontology in this work is similar to the role of the global conceptual schemas proposed in the works of Batini et al.[2] and Hayne et al. [9]. However, it is important to clarify that the ontology is in a higher abstraction level than global conceptual schemas. Each one of the conceptual schemas to be integrated is faced against the ontology, and for each conflict found the system calculates a similarity value.

The heterogeneities classification adopted in this work is the one defined by Visser et al. [23], which comprises the ones described by Bergamaschi [3] and Park [16]. In the scope of semantic heterogeneity, the ontological mismatching category is especially important, and is detailed below.

There are two basic types of ontological heterogeneity: conceptualization and explication [23]. Mismatch in terms of conceptualization happens between

two or more conceptualizations about a domain. They differ in terms of concepts (or entity, classes) covered or in how this concepts are related one to another. An explication mismatch is related on how the conceptualization is specified, that is, when two schemas have distinct definitions, but their terms, meanings or descriptions are the same.

As mentioned above, the conceptualization heterogeneity can be in respect of classes or relationships. In the first case, what happen is a conflict related to the classes distinguished in the conceptualization. This may be at the categorization level, which happens when the hierarchy of the same class is different in two schemas, because its subclasses are not the same. Also at the class level there are the disagreements in terms of aggregation, that is, the same concept can be designed in different levels of abstraction. A relation conflict exists in terms of the relationships between the concepts of different schemas. They can be structural, attributes names or attributes types. The structural conflict covers the associations between two or more concepts. The attributes conflicts can be of two types. One in respect of the attributes used to describe a concept (for example, a schema can have the attributes name and profession to the concept person, while another can have the attributes name, age and sex for the same concept person). The other type is in terms of the domain of the attributes.

4. The Methodology of the Ontology

The algorithm described next and shown in Figure 2 details in a high abstraction level the steps sequence to search and update the ontology.

The algorithm is semi-automate, because of the already discussed issues that make the process complete automate impossible. To minimize the need of the expert intervention two parameters have to be set at the beginning of the algorithm execution: The minimum and maximum accepted probabilities. The minimum probability is useful to filter some candidates. Only the concepts having the similarity probability higher than the minimum specified are shown to the expert. The ones with similarity probability lower than the minimum threshold are ignored. If none candidates reach the threshold, the input concept is considered as not existing in the ontology and added in it as a new concept. The maximum probability is used to make the selection of synonyms more automate. If one ore more of the ontology candidates have similarity probability higher than the maximum specified by the user, the one with the higher value is considered as equivalent of the input concept.

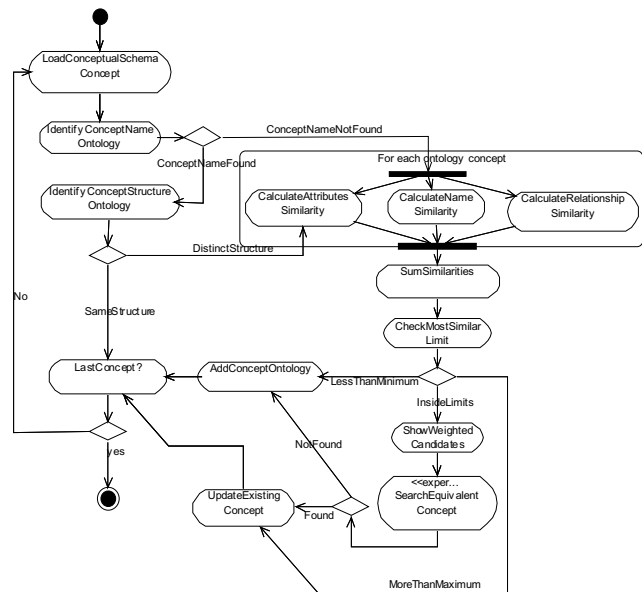


Figure 2 - The search and update algorithm

To guarantee the well working of the algorithm, it is necessary that every input conceptual schema has a metadata, specifying in which language the modeling is based.

Step 0 – Schema translation to the ontology’s language: If the ontology’s language is not the same of the one indicated by the conceptual schema’s metadata, this has to be translated, aided by a dictionary.

Step 1 – Search concept’s name in the ontology: If the concept’s name or one of its synonyms or acronyms (abbreviations) is found in the ontology, go to step 2. Else, if the term which nominates the concept is not found, go to steps 4, 5 and 6, in parallel.

Step 2 – Search concept’s structure in the ontology: Once the term which nominates the concept is found in the ontology, its structure is compared against the ontology, attribute by attribute. The algorithm verifies if every one of the input concept’s attributes exists in its correspondent concept of the ontology. In case of all the structure is equal in the ontology and in the input concept, go to step 3. If there are differences in at least one of the concept’s attribute, go to step 5.

Step 3 – Tests if it is the last concept: Search if the concept being processed is the last one of the input conceptual schema. If it is the last one, go to step the end. If there are more concepts go back to step 1 to processes of the next concept.

Step 4 – Calculate the similarity of the term that nominates the concept: The similarity between the input concept’s name and all the concepts names in the ontology is calculated. Go to step 7.

Step 5 – Calculate concept’s structural similarity: The input concept’s structural similarity is calculated, in terms

of its attributes. This comparison is performed against each one of the ontology's concept. Go to step 7.

Step 6 – Calculate relationship similarity: The input concept's relationship similarity is calculated, in terms of aggregation and composition associations, and also in terms of taxonomic (IS-A) relations. This comparison is performed against each one of the ontology's concept. Go to step 7.

Step 7 – Sum of the similarities: Based on some method of balance, the structural similarity, the name similarity and the relationship similarity of the input concept are summed, resulting in the similarity probability. This calculus is made for each ontology's concept. Go to step 8.

Step 8 – Verify threshold: Get the ontology's concept most similar to the input concept, and check its probability similarity values. If it is lower than the minimum limit chosen by the user, the input concept does not exist in the ontology, and then go to step 12. If its similarity probability is higher than the maximum threshold value specified by the user, the concept is chosen as a synonym of the ontology's correspondent concept, and go to step 11. If the similarity probability is between the maximum and minimum values then go to step 9.

Step 9 – Show candidates: Present each found candidates, with its balanced similarity probability. They are displayed ordered, with the ones with higher similarity first. Only the ones with similarity probability higher than the minimum threshold are shown. Go to step 10.

Step 10 – Term selection: At this point the domain expert intervention is necessary. He (or she) selects the concept he (or she) judges as the most adequate to represent the input schema's concept. If an ontology's existing concept is selected to represent the input schema's concept, go to step 11. If the expert decides that the input concept has not an equivalent in the ontology, and thus has to be added to it, go to step 12.

Step 11 – Update of an existing concept: Depending on from where this step was called, a distinct action is performed, to update the ontology. This action can be the addition of a new synonym or acronym to an existing term, the addition of a new attribute to an existing concept's structure, or the creation of a new relationship between two existing concepts. Go back to step 3.

Step 12 – Addition of a new concept to the ontology: A new concept is added in the on the ontology, with all its attributes. Go back to step 3.

4.1. Complementary Techniques

The use of ontology by itself does not provide a complete solution to the semantic integration problem. It is impossible to the ontology to contemplate all the ways to express a real world phenomenon. This happens because

of the inherent restrictions to the ontology and because of the differences derived by the individual process of interpretation of the reality [21]. Depending on the designer's geographical location the names for the same concepts may vary (for example, color and colour). Also the case of the acronyms has to be handled, that is, the way a term is written is not always the same, especially in the conceptual model, where the use of abbreviations is very common.

The human intervention in the resolution of the conflict is practically mandatory in the identification of correspondences process between different schemas. At most, what can be reached is that the ontology suggests the best solutions based on similarity and probability calculus [6]. To minimize the need of interaction with the domain expert, two things can be done. The first one is the investigation and implementation of similarity matching techniques [4]. This matching has to be made both in the concept naming level and in the structural level, addressing hierarchies, relationships and attributes of the ontology's existing concepts [3] [12].

The second part of the solution is about the update of the ontology in an "online" mode. Each time a concept is searched in the ontology and is not found, the expert has to select, among the shown candidates which one is the synonym of the input concept, or if it is a new concept that has to be added to the ontology.

This process not only semi-automate the process of conceptual schemas integration, but also offers the possibility to make the knowledge base richer, by adding new concepts.

4.2. The Similarity Calculus

To calculate the similarity between two concepts, one from the input conceptual schema and the other from the ontology, we adopt a hybrid approach, combining syntactic matching between strings and semantic matching.

In the syntactic matching, a distance function is applied over a pair of strings, to determine the dissimilarity between them. The smaller is this dissimilarity (measured by a integer value), the more similar are the strings [5].

In this work we adopted the Levenshtein distance, which is given by the number of changes we have to do in one string (insertions, deletion and substitutions) to make it equal to the other compared string. It is applied to the calculus of similarity between concept names (SimName(Cc,Co)) and attributes names.

The techniques to calculate the distance between two strings use only the syntactic features of the compared strings. They can be applied to acronyms and typing error cases [14], but none semantic is considered in these functions. Thus, for a correct semantic unification of

concepts, they have to be accomplished by some techniques capable to detect synonyms and to consider the context where the concept is in.

Our approach consider two semantic techniques to compare two concepts. The first one is the nearest neighbor [12], which is used to calculate the similarity in terms of attributes each concept has, and is given by the formula:

$$\text{SimAt}(Cc, Co) = \sum_{i=1}^n f(Cc_i, Co_i) \times \text{Wat}_i$$

where Cc and Co are, respectively, the conceptual schema's concept and the ontology's concept, n is the number of attributes considered, i is the index of the attribute being processed, f(Cc_i, Co_i) is the distance function between the attributes of the compared concepts (Levenshtein as proposed) and Wati is the weight of the attribute in the ontology.

The weight of an attribute is given by an adapted TF-IDF [4] formula:

$$\text{Wat} = 1 - (C_a/C)$$

where Ca is the number of concepts that have the attribute, and C is the total number of concepts. As can be deduced, the more concepts have the same attribute, the less significant this attribute is.

For the similarity between concept's relationships, three types are considered. The first one is the taxonomic (IS-A) associations, and the others two are the aggregation and composition ones. The similarity in terms of hierarchy is done by the formula:

$$\text{SimHier}(Cc, Co) = \frac{(\sum \text{Hier}(Cc, Pc) \cdot \text{Wt}(c, p))}{\text{NHier}(Cc, Pc)}$$

where Hier(Cc, Pc) is each one of the taxonomic relationships existing in both the conceptual schema and in the ontology. Wt(c, p) is the weight of the hierarchical relationship arc and Nhier(Cc, Pc) is the number of IS-A associations in both the ontology and the conceptual schema.

The weight Wt(c, p) of an taxonomic arc is given by the following formula [20][13]:

$$\text{Wt}(c, p) = \frac{(E) \cdot (d(p)+1) \cdot (IC(c) - IC(p))}{E(p) \cdot d(p)}$$

where E is the d(p) is the depth of the parent node (p) of the node corresponding to the concept being compared. E is the density of the whole ontology's hierarchy, that is, the number of nodes it has. E(p) is the density of the taxonomy considering the node p as the root concept, that is, the number of direct and indirect children it has. Finally, IC is the information content of the node. IC represents the amount of information the node has [19], and its value is given by:

$$IC(c) = -\log((\sum (1/\text{sup}(c))) \cdot 1/N)$$

where sup(c) is the number of super classes (direct or indirect) the class c has, and N is the total number of concepts of the ontology. As it can be deduced, the more specialized a concept is, the more information it has.

At last, the aggregation and compositions links are considered to calculate the similarity of two concepts, by the simple formula:

$$\text{SimRel}(Cc, Co) = (\sum \text{Rel}(Cc, Co)) / \text{Rel}(Cc)$$

where Rel(Cc, Co) is each composition/aggregation link existing both in the ontology and in the conceptual schema and Rel(Cc) is the ones present only in the conceptual schema.

The final value of similarity is given by a balanced sum of the similarities:

$$\text{Sim}(Cc, Co) = \text{WN} \cdot \text{SimName}(Cc, Co) + \text{WA} \cdot \text{SimAt}(Cc, Co) + \text{WH} \cdot \text{SimHier}(Cc, Co) + \text{WR} \cdot \text{SimRel}(Cc, Co)$$

where WN, WA, WH and WR are the weights of names, attributes, hierarchies and relationships similarities.

5. conclusions

Ontology can contribute in a significant way in the conceptual modeling of GDB. Because of the absence of a standard data model, the interchange of GDB schemas is a difficult task. Thus, the ontology can be used, associated to a canonical data model, to help not only the interchange of the schemas, but also the understanding of it and avoiding conflicts, such as heterogeneities and redundancy.

Although our proposal is focused in geographic databases conceptual schemas, this methodology may be applied for the unification of any kind of database conceptual schemas.

To continue the research developed in this paper, some future works have to be done. One is the development of formal methods to model the correspondences and transformation from the ontology to some specific conceptual models, and from these conceptual models to a canonical data model.

Other important future work is the implementation of the algorithm proposed in this paper, applying the similarity matching algorithms to balance the similarity probabilities between concept's terms and attributes from the input conceptual schema and the ontology's concepts.

References

- [1] G. Bassalo, C. Iochpe, N. Bigolin. "Representing schemas in the attribute-value format for the inference of analysis patterns" In proc. Of the IV Brazilian Symposium on GeoInformatics (GeoInfo), 2002 (in portuguese).
- [2] C. Batini, M. Lenzerini, S. Navathe. "A Comparative Analysis Of Methodologies For Database Schema Integration". In ACM Computing Surveys, v.18, n.4, p.323-364, 1986.
- [3] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, S. Vincini. "An Intelligent Approach to Information Integration." In Internation Conference on Formal Ontology in Information Systems (FOIS'98), 1998.
- [4] W. Cohen. "Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Text Similarity", In Proceedings of the 1998 ACM SIGMOD international conference on Management of data, 1998.
- [5] W. Cohen, P. Ravikumar, S. Fienberg. "A comparison of String Metrics for Matching Names and Records". In Proc of IJCAI 2003 – Workshop on Information Integration on the Web. Acapulco, Mexico, 2003.
- [6] Z. Cui, D. Jones, P. O'Brien. "Semantic B2B Integration: Issues in Ontology-based Approaches", In Sigmod record web edition, v.31, 2002.
- [7] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth. "From Data Mining to Knowledge Discovery in Databases." In AI Magazine, v.17, n.3,p.37-54, 1996.
- [8] H. Gamma, R. Johnson, J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software." Addison-Wesley, 1995.
- [9] S. Hayne, S. Ram. "Multi-User View Integration System(MUVIS): An Expert System for View Integration." In proc. Of the 6th International Conference on Data Engineering, p. 402-409, 1990.
- [10] G. Hess, C. Iochpe. "Using the GML for the Identification of GDB Analysis Patterns Candidates." In proc. Of the V BrazilianSymposium on GeoInformatics (GeoInfo), 2003 (in portuguese).
- [11] G. Hodge. "Knowledge Organization Systems: An Overview." In System of knowledge Organization for Digital Libraries: Beyond Traditional authority files, 2000.
- [12] A. Holt. "Understanding environment and geographical complexities trough similarity matching". In Complexity International, number 7, 2000.
- [13] J. Jiang, D. Conrath. "Semantic Similarity Based in Corpus Statistics and Lexical Taxonomy". In Proc of International Conference Reasearch in Computational Linguistics (ROCLING X). Taiwan, 1997.
- [14] A. Monge, C. Elkan. "The field matching problem: Algorithms and Applications". In Proc of the Second International Conference and Data Mining. 1996.
- [15] OpenGIS Consortium. "Geography Markup Language (GML) 3.0". Open GIS Implementation Specification, 2003. Available in <http://www.opengis.net>. Last access in december 2003.
- [16] J. Park. "Schema Integration Methodology and Toolkit for Heterogeneous and Distributed Geographic Databases". working paper, 2001.
- [17] C. Partridge. "The Role of Ontology in Integrating Semantically Heterogeneous Databases". Technical Report, 2002.
- [18] J. Qin, S. Paling. "Converting a controlled vocabulary into an ontology: the case of GEM." In Information Research 6, 2001.
- [19] P. Resnik. "Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language". Journal of Artificial Intelligence Research, number 11, p. 95-130, 1998.
- [20] R. Richardson, A. Smeaton, J. Murphy. "Using WordNet as a Knowledge Base for Measuring Semantic Similarity between Words". In Proc. of the AICS Conference. Dublin, Ireland, 1994.
- [21] A. Sheth. "Changing focus on interoperability in information systems: From systems, syntax, structure to semantics." In Interoperating Geographic Information Systems, 2000.
- [22] C. Silva, C. Iochpe, P. Engel. "Using Knowledge Discovery in Database to Identify Analysis Patterns." In Proc. Of the 5th International Conference on Enterprise Information System (ICEIS), 2003.
- [23] P. Visser, D. Jones, T. Bench-Capon, M. Shave. "An Analysis of Ontology Mismatches Heterogeneity versus Interoperability". In proc. Of the AAAI 1997 Spring Symposium on Ontological Engineering, 1997

Automated Risk Assessment for Managing Software Projects

B. Ray, T. Klinger, R. Delmonico and P. Santhanam

{bonnier, tklinger, rmd, pasanath} @us.ibm.com

Center for Software Engineering, IBM Thomas J. Watson Research Center

19 Skyline Drive, Hawthorne, NY 10532

Abstract. Managing risk in software projects remains a significant challenge. To meet this challenge, software development organizations collect a broad range of development data and metrics such as change requests, defect information, status of test cases, and others. In this paper, we examine some of the published industry best practices for in-process software project assessment and extract a few common types of analysis. We describe a prototype decision support tool which implements some of these practices using rules and statistical analysis. A key aspect of the tool is the use of time series based analysis, which measures the evolution of software through the development process. We assess the applicability of this approach in practice by comparing the output of the tool against manual assessment of actual project defect data. This work shows the feasibility of establishing an automated risk management framework based on a realistic set of metrics and analyses from a practical software engineering perspective.

Keywords: software risk assessment, in-process metrics, rule based, statistics, automation.

1. Introduction

The success of a software project depends on the ability of a development organization to deliver software that meets the needs of its customers, on time and with acceptable quality. An integral aspect of the software development process [1] is the use of software metrics [2] to do risk management. Pfleeger et al.[3] pointed out the existing large gap between the output of the research community in software metrics and the actual use of the metrics by the practitioners for real product management. Given the schedule and resource pressures in a typical software development organization, there needs to be a more automated way to integrate metrics based risk assessment into the normal development process.

Software risk management can target long or short term issues. The longer term issues address improvements to the software for a better stability and maintainability of a product over many releases. This may involve actions such as reducing complexity by refactoring code, identifying and rewriting error prone modules, etc.[4,5] or targeting development process maturity improvements such as the

Capability Maturity Model [1]. On the other hand, short term risk management is focused solely on assessing and managing risk as the software progresses through the various check points of the current development cycle. In this paper we consider the problem of risk management from only this short term perspective.

Section 2 briefly reviews some of the published best practices in the industry. Section 3 describes how these best practices can be mapped to a few types of analysis techniques. Section 4 describes our rule language and the architecture of an automated decision support tool. Section 5 illustrates the application of this tool against available defect data to evaluate the readiness of a project to exit function test phase, and compares the results against a manual assessment. Section 6 provides conclusions. The scope of the paper is to show the feasibility of establishing an automated risk management framework based on a realistic set of metrics and analyses. The results of experiences of using this framework in actual projects will be addressed in a future paper.

2. Best Practices for In-Process Risk Management

Over the years, there have been a number of papers addressing the use of software metrics for in-process project management. Generally this body of work focuses on a specific set of metrics to monitor various aspects of risk during the execution of a project. The data could then be used to assess the status of the software at the appropriate checkpoints during development or to decide on the suitability of the software for release to the customers. In this section we briefly review a representative subset [6-12] of this literature.

Daskalantonakis [6] described the use of Goal/Question/ Metrics approach with explicit application to longer term process improvements vs. in-process project control. Stark et al. [7] discussed an innovative use of metrics to decide on various key issues for topics such as project replans, maintenance or redesign of software, integration of subsystems, assessment of testing schedule. Foody [8] gave specific recommendations how to decide when software is ready for release based on a set of criteria involving coverage of functionality, defect severity

considerations, defect discovery rate, and so on. Ebert [9] discussed the concept of “technical controlling” to address software process analysis and improvement. Kan et al. [10] describe their use of in-process test metrics such as test progress S-curve, defect discovery rates, defect backlog over time, product size over time, etc. Bassin et al. [11,12] described the use of metrics based on test case execution results in combination with test case information or defect data captured using Orthogonal Defect Classification (ODC) [13,14]. Table 1 catalogs some of the prevalent metrics in the published literature [6-12] along the key focus areas of analysis, such as the progress of the project against the schedule, the stability of the product and the effectiveness of the defect removal activities. Metrics in the second column are meant to be representative and not exhaustive.

3. Core Analysis Types

Although the metrics represented in Table 1 span different kinds of data (e.g. Defects, Test cases, etc.) and focus areas, they can be assessed using four basic analysis types:

1. *Comparison of distributions*: Does the distribution of one variable across a set of categories match that of another variable?

2. *Comparison to a constant*: Is the value of a variable above/below a specified threshold value?
3. *Comparison to a variable*: Is the value of one variable above/below that of another variable?
4. *Trend analysis*: How is a variable evolving over time?

Table 2 shows a mapping of each of the metrics of Table 1, with the corresponding analysis type that could be used to evaluate the metric. More complex analyses can be obtained by applying a combination of these types to address a particular concern. We note that each type of analysis can also be implemented in several ways.

3.1 Comparison of distributions

Two distributions can be compared by simply computing the difference in the percentage of variable values falling into each category for the two different variables, with a difference declared if one or more of these differences is more (less) than zero or another specified threshold. Alternatively, the data used for analysis can be considered as a random sample from an underlying population, and statistical techniques used to compare the distributions. Different formulations of a Chi-squared goodness-of-fit (GOF) test can be used to compare an observed distribution to a target distribution or to compare one observed distribution to another observed distribution. See, for example, Section 13.3 of [15]. The statistically

Table 1: A sample list of metrics and criteria for managing in-process development from references [6-12]

Focus Area	Metrics Used (with References)
Schedule Integrity	1. Planned test case progress, Attempted test case progress, Successful test case progress over time in weeks.[9,10,12] 2. Intended test effort vs. Actual test cases by ODC Triggers [11] 3. Intended ODC Trigger distribution by Activity vs. Actual ODC Trigger distribution by Activity [12]
Product Stability	4. Cumulative number of requirements compared to cumulative number of requirements implemented in design over time [6] 5. Predicted number of defects (derived from models or prior releases) and compared to the actual found in the project over time in weeks. [6,10] 6. Severity of Open problems over time [6] 7. All Severity 1 and 2 problems be closed and only limited numbers of Sev.3 and Sev 4 problems open. [6,8,12] 8. Defect backlog over time [10] 9. Release size over time [10] 10. ODC Defect Type over time [12] 11. ODC Defect Type vs. ODC Qualifier [12]
Test Effectiveness	12. Number of defects/kLOC or currently open number of defects/kLOC vs. Number of active test hours per kLOC. [7] 13. Defect arrival rate is less than 40 defects per 1000 test hours.[8] 14. Full regression suite covering 100% functionality, 80% branch coverage and 100% of the procedures. [8] 15. ODC Triggers over time [12]

Table 2: Mapping of the analysis types to the metrics from Table 1.

Analysis Type	Metrics from Table 1
Comparison of distributions	2,3
Comparison of variable to constant	5,7,11,13,14
Comparison of variable to variable	4,5,12
Trend Analysis	1,6,8,9,10,15

based comparison allows for observed variations between distributions that depend on the underlying sample size.

3.2 Comparison to a Constant

Comparison to a Constant and Comparison to a Variable can be accomplished by simply summarizing relevant data and checking whether the summarized variable takes values less than (greater than) a specified constant or another expression. Comparison to a Constant can also be framed in a probabilistic context, with statistical techniques used to determine whether the observed data are consistent with a hypothesized threshold value. For instance, a binomial test of proportion ([16, Section 6.5]) is an appropriate way to assess the chance of obtaining the *observed* percentage of items, assuming that the underlying population contains a specified percentage of these items. A *t*-test ([16, Section 6.3]) could be used to assess whether the observed sample average of a variable is consistent with a specified average value for the variable in the population. Other statistical methods could also be used, depending on the assumptions imposed on the data.

3.3 Comparison to a Variable

The Comparison to a Variable type allows rules that target changes in volumes or percentages between

specified time frames. For instance, rules that focus on the size of the difference in percentage of defects of a certain type between adjacent periods (e.g. every week) can be formulated in this framework. Thus this type of analysis is useful not only for exit evaluations, but also for on-going monitoring of progress.

3.4 Trend Analysis

This is one of the most important type of analysis deployed in managing a software project. The particular challenge of the analysis is the selection of appropriate subset of the data for assessing trends such as the last two months of a six month project. Several alternatives exist for trend analysis, depending on whether it is desired to test for existence of a positive (negative) trend or fit a (linear) trend to observed data, and again depending on data assumptions. Table 3 shows the specific types of analysis chosen for our implementation.

Note that other authors, for example [17], have discussed the use of probabilistic methods as an aid for decision making in the software development process. The aim of the current paper is somewhat different, however, in that we show how a small set of analysis types can cover a broad range of industry best practices, and indicate how these types can be embodied in a rule-based engine described in the next section to provide consistent assessments useful for decision support.

4. Rule Language and Implementation

We have developed a declarative language to express rules for the classification of risk using the analyses described in Section 3. Figure 1 shows a sample rule in this language which uses both a trend analysis and a variable-to-constant comparison to classify Severity 2 defects. This rule is taken from a ruleset for assessing risk according to defect severity (see Section 5 for a detailed sample analysis using this rule).

In English, the rule says:

IF the trend in the percentage of Severity 2 defects over the last ¾ of the time periods supplied is not decreasing

Table 3: Implementation details for the analysis types.

Analysis Type	Implementation
Comparison of Distributions	1. Chi-squared GOF for comparing a single population to a target 2. Chi-squared GOF for comparing two populations
Comparison to Constant	1. Deterministic 2. Binomial test of specified proportion
Comparison to Variable	1. Deterministic
Trend Assessment	1. Cox and Stuart test for trend , see ref. [18] Section 3.5 2. Linear regression

Rule name: Trend and percent of Severity 2 defects

Precondition:

trend(sev2pct, LAST_THREE_QTRS) != DECREASING
OR
pct(sum(sev2, LAST_QTR), sum(tvot, LAST_QTR)) >=20

Classification: Risk = Medium

Figure 1: A typical rule

OR the percentage of the total number of Severity 2 defects is ≥ 20 .
THEN the risk is Medium.

Here *sev2pct* is a timeseries variable representing the percentage of Severity 2 defects at each period in the timeframe, and *sev2* and *tot* represent to the timeseries containing the number of Severity 2 defects and the total number of defects at each period, respectively.

In general, each rule has three parts: the *name*, the *precondition* and the *classification*. When the precondition of a rule is true then the rule asserts its classification for a given set of defect data (otherwise it makes no assertion about the data). Individual rules like the one shown can be combined into *RuleSets*. A RuleSet is evaluated by evaluating its constituent rules and summarizing their results into a single classification of the data. The order of evaluation is not important since the rules are all independent. The summarization step is necessary since the rules may not yield mutually consistent risk assessments. For example, looking only at the number of Severity 2 defects using the rule in the figure, we may believe that the risk is Medium. However, another rule focusing on the total number of defects may assert that the risk is high. These results must somehow be reconciled. Currently we offer two summarization algorithms to address this problem: *worst case* (the worst or most risky classification from any rule is the classification of the data) and *mode* (the most frequent classification is chosen). Others may easily be added to perform more complex summarizations. So far, we have only used worst case summarization in our analyses.

The precondition of a rule can be any boolean expression using logical operators AND, OR, NOT and the

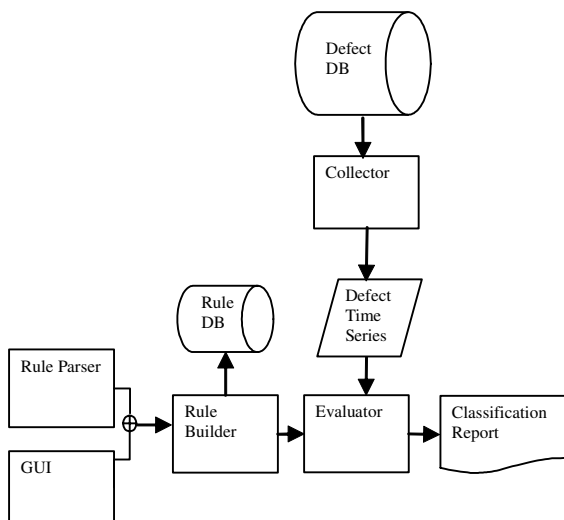


Figure 2: System architecture

relational operators. Terms are either literals (Boolean, String, Real, Integer, Enumerated, TimeFrame), variables of type TimeSeries, functions on simple types, or functions on TimeSeries variables. The TimeFrame literals are a set of predetermined relative time frames including for example LAST_PERIOD, FIRST_HALF, etc. Functions currently supported are:

- **trend**(TimeSeries *ts*, TimeFrame *tf*) : TrendDirection
Returns whether the TimeSeries *ts* is INCREASING, DECREASING, or UNCHANGED (the three values in the enumerated type TrendDirection) over *tf*.
- **sum**(TimeSeries *ts*, TimeFrame *tf*) : Integer
Returns the sum of the values in *ts* over *tf*.
- **pct**(Real *r1*, Real *r2*) : Real
Returns $100 * r1 / r2$.
- **probProportionTest**(TimeSeries *tsObs*, TimeSeries *tsTot*, TimeFrame *tf*, Integer *threshold*, boolean *aboveOrBelow*) : Boolean
Returns true iff the proportion of the sum of the observed timeseries values *tsObs* to the sum of the totals *tsTot* is above/below *threshold* with statistical significance.
- **linearTrendTest**(TimeSeries *ts*, TimeFrame *tf*, Integer *threshold*, Integer *periodOffset*, Boolean *aboveOrBelow*) : Boolean
Returns true iff the linear extrapolation of TimeSeries *ts* for *periodOffset* periods past TimeFrame *tf* is above/below *threshold*.

Of the analysis types discussed in section 3, these functions support: comparison to a constant/variable (both deterministic and probabilistic) and trend analysis. Distribution comparison is currently implemented, but not integrated with the rules engine.

The analyses and rule language are fully implemented in Java and are being deployed as part of an internal IBM decision support tool for software risk management. Figure 2 shows a high-level view of the system architecture.

Reading from the left of this diagram, there are two choices for entering rules into the system. A text parser allows rules to be entered in a language similar to that used in Figure 1. However, a Graphic User Interface front-end is the typical means of entry for the user. Both make use of a Rule Builder component to assemble an Abstract Syntax Tree (AST) representation, which can be persisted to a file or database. The defect data is collected into a time series representation which, together with the rule set, can be evaluated to yield the risk classification report.

5. A Comparison of Manual and Automated Analyses

In this section we present an analysis typically performed by a person for assessing function test exit readiness and describe the corresponding ruleset for

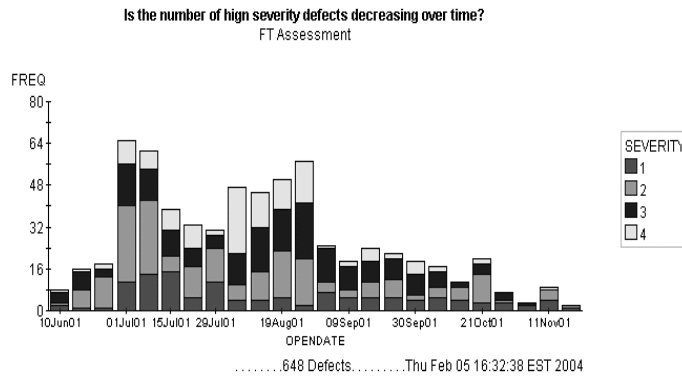


Figure 3: Distribution of defects by Severity over Time

automated assessment. Our goal here is simply to illustrate how the manual assessment can be adequately captured in our rule language, and not to argue for the validity of that assessment.

Our analyst typically begins with a visual assessment of the data. For example, Figure 3 shows a chart of number of defects over time broken down by defect severity. By “eyeballing” the shape of the defect curves over time in each severity group, the analyst classifies the trends as increasing, decreasing or unchanged. The analyst then consults a flowchart such as that shown in Figure 4 to determine the overall assessment of the project. Although not explicitly shown in Figure 4, the analyst also takes into consideration other factors such as volume of defects and the specific customer impact [13,14] of the defect in assessing potential risk from high severity defects. For instance, if the volume of Severity 1 defects is decreasing over time, but is still above one or two, the product stability is considered to be at risk. It is important to note that other aspects of manual analysis may include some implicit smoothing/filtering in the visual trend analysis, e.g., omitting data for the first few weeks of the test period.

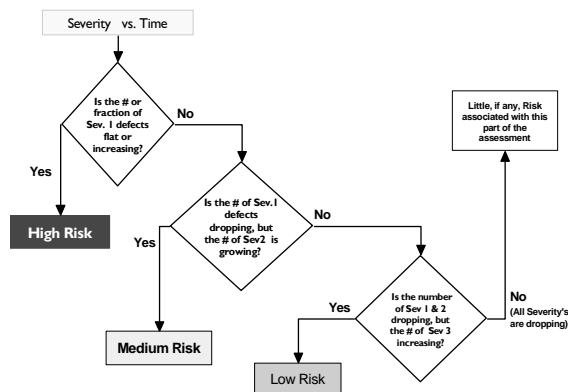


Figure 4: Flow diagram capturing the analysis of Severity over Time.

Table 4: Manual risk assessment based on Fig. 3

1. The volume of Sev 1 defects in most recent time periods indicates that the product does not yet appear to be stable, although the potential impacts of these defects on the customer is low.
2. Consistent surfacing of Sev 1 defects indicates potential product instability.
3. The volume, % or trend of Sev 2 defects in most recent time periods indicates there are still too many high severity defects, although the potential impact of these defects on the customer is low.
4. An increase in Sev 4 defects relative to Sev 3 defects over time is desirable.

These details are not captured formally so that there is a possibility for subjectivity and inconsistency in the conclusions from time to time. The analyst’s risk assessment of the data shown in Figure 3 is summarized in Table 4.

To capture the analyst’s assessment, we formalized the flowchart of Figure 4 into an initial rule set, translating imprecise statements like “trend is growing” into more precise statements in our rule language. We refined the ruleset by analyzing it on historical data and adjusting it when it differed in its classification from the analyst. The end product was the ruleset shown in Table 5. We present the rules in English for readability but they are represented internally in the form described in Section 4. The automated assessment of the sample dataset of Figure 3 indicated potential for medium risk. This was based on the satisfaction of the preconditions of rules 2, 3, 5 and 7 in Table 5 under the worst case summarization policy. This corroborates the observations by the analyst in Table 4.

An automated assessment provides a number of benefits over manual analysis. It allows consistent and repeatable evaluation of project data. This supports enforcement of organizational policies and normalization of the analysis both within a project at different times and between different projects. In addition it provides the ability to quickly summarize, compare, and trend data across multiple attributes, filtered by different time frames and other criteria. For example, our current implementation to perform a function test exit evaluation consists of 47 separate rules involving 6 timeseries variables and 9 different time frames. With the tool these rules were evaluated in seconds against real project data; without the tool a similar evaluation would be so onerous as to make it essentially impossible in practice. Statistical computations performed in the checking contribute to the sophistication of the evaluation.

Table 5: Ruleset for automated assessment

1. If the total volume of Severity 1 defects is greater than 2 in the last ¼ of the specified time frame and more than 50% of these defects are high impact defects, then risk is High.
2. If the total volume of Severity 1 defects is greater than 2 in the last ¼ of the specified time frame but less than 50% of these defects are high impact defects, then risk is Medium.
3. If one or more Severity 1 defects occur in each ¼ of the specified time frame (i.e., consistent surfacing of Severity 1 defects), then risk is Medium.
4. If the trend in Severity 2 defects is not decreasing in the last ¾ of the time frame or there are more than 20% Severity 2 defects in the last ¼ of time frame, and more than 50% of Sev 2 defects found in last ¼ of time frame are high impact, then risk is Medium.
5. If the trend in Severity 2 defects is not decreasing in the last ¾ of the time frame or there are more than 20% Severity 2 defects in the last ¼ of the time frame, but less than 50% are high impact, then risk is Low.
6. If the trend in Severity 3 defects is not decreasing in the last ½ of the time frame or there are more than 40% Severity 3 defects in the last ¼ of the time frame, then no risk specification is given, but user is informed of status.
7. If the percentage of Severity 3 defects is more than the percentage of Severity 4 defects in the last ¼ of time frame, then no risk specification is given, but user is informed of status.

6. Conclusions

In this paper we have presented an automated risk assessment framework and described its implementation. We have shown how a representative sample of the industry best practices in software metrics can be accommodated in this framework by categorizing the underlying methods into a few types of analysis implemented in our system. To illustrate the applicability of our approach to a typical assessment problem, we compared manual and automated analysis on actual function test exit evaluation data. Although the examples considered focused on defect-based monitoring, the system is applicable to any set of project metrics used for monitoring progress, such as test case completions, backlog, and so on. We plan to discuss the results of deploying this system in software projects across IBM in a future publication.

We thank Kathryn Bassin and Theresa Kratschmer for serving as experts in defect analysis.

7. References

- [1] Watts S. Humphrey, "Managing the Software Process", Addison Wesley, Reading, MA, USA, 1990.
- [2] R. B. Grady and D. L. Caswell, "Software Metrics: Establishing a Company-wide Program", Prentice-Hall, Inc., Englewood Cliffs, NJ, 1987.
- [3] S. L. Pfleeger, R. Jeffrey, B. Curtis and B. Ketchenbaum, "Status Report on Software Measurement", IEEE Software, Vol.14, No.2, 1997 pp.33-43.
- [4] W. M. Evancho and R. Lacovara, "A Model Based Framework for the Integration of Software Metrics", J. Systems and Software, Vol.26, 1994, pp 77-86.
- [5] J. P. Hudepohl, S. J. Aud, T. M. Khoshgoftaar, E. B. Allen, and J. Mayrand, "Emerald: Software Metrics and Models on the Desktop", IEEE Software, Vol. 13, No. 5, 1996, pp.56-60.
- [6] M. K. Daskalantonakis, "A Practical View of Software Measurement and Implementation Experiences Within Motorola", IEEE Trans. Software Engineering, Vol. 18, No. 11, 1992, pp.998-1010.
- [7] G. Stark, R.C. Durst, and C.W. Vowell, "Using Metrics in Management Decision Making", Computer Vol. 27, No.9, 1994, pp.42-48.
- [8] M. A. Foody, "When is Software Ready for Release?", Unix Review, March 1995, pp.35-41.
- [9] C. Ebert, "Technical Controlling and Software Process Improvement", J. Systems and Software, Vol.46, 1999, pp. 25-39.
- [10] S. H. Kan, J. Parrish and D. Manlove, "In process Metrics for Software Testing", IBM Systems Journal, Vol.40, No.1, 2001, pp. 220-241.
- [11] K. Bassin, S. Biyani, and P. Santhanam, "Evaluating the Software Test Strategy for the 2000 Sydney Olympics", Proceedings of the IEEE 12th International Symposium on Software Reliability Engineering, 2001, pp.264-273.
- [12] K. Bassin, S. Biyani, and P. Santhanam, "Metrics to Evaluate Vendor-developed Software Based on Test Execution Results", IBM Systems Journal, Vol. 41, No.1, 2002, pp.13-30.
- [13] R. Chillarege, I.S. Bhandari, J. K. Chaar, D. S. Moebus, B. K. Ray, and M.-Y. Wong, "Orthogonal Defect Classification: A Concept for In-Process Measurements", IEEE Trans. Software Engineering, Vol.18, No.11, 1992.
- [14] K. Bassin, T. Kratschmer, and P. Santhanam, "Evaluating Software Development Objectively", IEEE Software, Vol. 15, No.6, 1998, pp.66-74.
- [15] J. Rice, "Mathematical Statistics and Data Analysis", Duxbury Press, Belmont, CA, USA, 1995.
- [16] S. B. Vardeman, "Statistics for Engineering Problem Solving", PWS Publishing Company, Boston, MA, USA, 1994.
- [17] I. Rus, S. Biffi, and M. Halling, "Systematically Combining Process Simulation and Empirical Data in Support of Decision Analysis in Software Development", Proceedings of the SEKE 2002, pp. 827-833.
- [18] W. J. Conover, "Practical Nonparametric Statistics, 2ed", John Wiley & Sons, Inc., New York, NY, USA, 1980.

Clarifying the Relationship between Software Architecture and Usability

Natalia Juristo, Ana M. Moreno
School of Computing - Universidad
Politécnica de Madrid, Spain
natalia@fi.upm.es, ammoreno@fi.upm.es

Maria Isabel Sánchez
School of Computing - Universidad Carlos III
de Madrid, Spain
misanche@inf.uc3m.es

Abstract

This paper examines in a problem posed recently concerning the relationship between software system usability and architecture. Here, we try to empirically clarify this relationship, focusing on the concept of architecture-sensitive usability mechanism. This concept represents specific usability issues that can improve software usability and that have demonstrated architectural implications. Accordingly, this paper outlines how usability needs to be decomposed to be dealt with from an architectural point of view and how the architecture-sensitive usability mechanism emerges. A list of architecture-sensitive usability mechanisms is presented and the procedure for outputting their respective architectural implications is discussed.

1. Introduction

Usability is an important component of software quality. Although there is no established set of critical software quality attributes, several classifications agree on the importance of considering usability as a quality attribute [1][2][3]. Additionally, usability is increasingly recognized as a quality attribute that has a big impact on software development [4].

To understand the depth and scope of the usability of a system, it is useful to make a distinction between the visible part of the user interface (buttons, pull-down menus, check-boxes, background color, etc.) and the interaction part of the system. By interaction we mean the coordination of information exchange between the user and the system. A system's usability deals not only with the user interface, but mainly with the user-system interaction. This interaction must be carefully designed and should be considered when designing not just the visible part of the user interface, but also the rest of the system. For example, the provision of continuous feedback for users is a primary usability feature, and its implementation needs to be considered when designing the system. System operations have to be designed so as to allow information to be frequently sent to the user interface to keep users informed about the current status

of the operation. So, although this information could be displayed by different means (percentage-completed bar, a clock, etc.) and these means are interface or presentation issues, the feedback feature is not just an interface aspect. It is a functionality that affects system usability and should be considered during design, as the design is affected by the decision on whether or not to include this usability feature.

However, seminal interactive system architectures, such as Model-View-Controller (MVC) and Presentation Abstraction Control (PAC) [5] seem to assume that usability only affects the presentation and dialogue components of an interactive application. Based on this assumption, these architectures decouple the application features from the user interface, such that each can be designed and modified more or less independently of the other. This assumption does not consider the fact that functionalities buried in the application logic can sometimes affect the usability of the whole system.

Recently, some groups have been working on identifying specific usability aspects with connections in the software architecture to try to clarify this relationship [6] [7]. These papers show how even if the presentation of a system is well designed, system usability can be greatly compromised if the underlying architecture and designs do not make the proper provisions for user concerns.

In this paper, our aim is to contribute to this clarification by empirically studying the relationship between software usability and software architecture¹. Note that it is important to clarify this relationship, because, as mentioned above, if any such relationship exists, developers should bear usability issues in mind when defining the overall system and not just when working on the user interface.

To deal with this relationship, we have decomposed usability into lower level concepts more related to the software solution. As we will see in section 2, these concepts are usability attributes and usability properties.

¹ The content of this paper is part of the research done in the STATUS project: European Union funded project IST-2001-32298.

Then the concept of architecture-sensitive usability mechanism is introduced in section 3, identifying specific usability features that will address a particular usability property and whose inclusion in a software system will have a specific effect on its architecture. Section 4 shows an example of the architectural implications for one such architecture-sensitive usability mechanism (Undo). It also describes the empirical process followed to identify these implications, and refers to the site where the implications of the other architecture-sensitive usability patterns identified can be found. From our research, we conclude that there is a relationship between usability and software architecture and that it is, therefore, dangerous to assume that usability will only affect the presentation component of our software systems. Usability also needs to be dealt with when designing the logic of applications.

2. Decomposing Usability from the Architectural Viewpoint

One of the problems of working with usability from a design perspective is that it is a broad and abstract concept that is hard to grasp. Therefore, the best way of addressing the concept of usability is to decompose it. The first level of the usability decomposition is what is called usability attributes in the (Human Computer Interaction) HCI field. Usability attributes are precise and measurable components of the abstract concept that is usability. Usability has been decomposed into attributes in the HCI field mainly for evaluation purposes. Although different authors have proposed different usability attribute classifications, the view that appears to be shared by most of the prominent authors in the field is that the main *usability attributes* are [11] [12] [14]:

- **Learnability**, which is composed of two complementary aspects: how quickly users can learn to use the system for the first time and how easy it is to remember how to operate the system after not having used it for some time.
- **Efficiency of use**, which refers to how efficiently the user performs a task using the system, that is, this attribute measures the efficiency of the software system used by the user. Note that this attribute is not the same as the classical quality attribute of efficiency, understood as system efficiency.
- **Reliability of use**. Again, this parameter is not to be confused with system reliability. It refers to the reliability of the user performing a task using the system. Therefore, this attribute refers to the errors made by the user when using the system, not the system errors.

- **Satisfaction** is the most subjective attribute and refers precisely to the user's subjective view of the system.

However, these usability attributes are very far removed from software design, that is, the effect that these attributes have on software architecture cannot be determined directly. Therefore, the approach that we have followed has been to decompose these attributes into intermediate levels of concepts that are increasingly closer to the software solution. The first one of these concepts is usability property.

We have identified *usability properties* from the HCI field. HCI researchers have defined some concrete aspects to help developers to build usable systems. Each author has named these tips differently: design heuristics [8], rules of usability [9], principles of usability [10][11], ergonomic principles [12], etc. We have compiled these design heuristics and principles that different authors suggest for developing more usable systems [8][9][10][11][12][13][14] and have arrived at the following usability properties for a software system:

- **Keeping the user informed**. The system should inform users at all times so that they know what is going on.
- **Error management**. The system should provide a way to manage errors. This can be done by error correction or error prevention.
- **Consistency**. The system should be consistent in all aspects of interaction, that is, in the interface and in the way we provide functionality.
- **Guidance**. We should provide informative, easy-to-use and relevant guidance and support both in the application and in the user manual to help the user understand and use the system.
- **Minimize cognitive load**. Systems should minimize the cognitive load, e.g., humans have cognitive limitations, and systems should bear these limitations in mind.
- **Explicit user control**. Users should feel that they are in control of the interaction.
- **Natural mapping**. The system should provide a clear relationship between what the user wants to do and the mechanism for doing it.
- **Ease of navigation**. Systems should be easy to navigate.
- **Accessibility**. Systems should be accessible in every way that is required. This property includes internationalization, multi-channeling and accessibility for disabled people.

Although this classification could contribute to somehow structuring the field of design heuristics, an important problem still remains to be addressed.

Usability properties may be useful as possible sources of requirements to be satisfied by a usable software system. However, developers have no systematic way of incorporating them into their developments. In other words, they need to know what particular elements a software system has to include to satisfy a usability property. Therefore, usability properties need to be further elaborated if we want developers use them to incorporate specific functionalities to improve the usability of the software systems.

3. Architecture-Sensitive Usability Mechanisms

Very recently, the HCI community has developed the concept of usability pattern. There are several a few lists of usability patterns, the most commonly referenced being the Amsterdam Collection [15] and Common Ground [16]. HCI usability patterns provide usability solutions (allow the user to undo at least the last couple of actions, provide feedback to the user every two seconds of command processing, in forms to be filled by users arrange the blanks in an order that makes sense semantically, use different colors to identify the major sections of the screen, etc.) to common problems.

Note that, on the one hand, the inclusion of some of these usability solutions in a software system will help to address specific usability properties. On the other, the inclusion of some of these solutions in a software system could have an effect on its software architecture and not only on its user interface.

So, we have developed the concept of *architecture-sensitive usability mechanism*, to refer to specific usability features that have an impact on the software architecture (as we will see in the next section) and address particular usability properties. In other words, we have descended another level in our approximation of usability to architectural design, defining the concept of architecture-sensitive usability mechanisms. An architecture-sensitive usability mechanism addresses a need identified by a usability property at the requirements stage and that has a specific effect on the design of the software system.

Note that we avoid to use the concept of usability pattern, as from a software engineering perspective, patterns should provide validated design solutions to repetitive problems [17], while, architecture-sensitive usability mechanisms represent usability features that affect software architecture. As noted at the end of this paper, we intend to pursue this work in the future by approximating these mechanisms to architectural sensitive usability patterns, adding to the usability solutions proposed by the HCI community particular design solutions.

Table 1 shows the relationship between usability properties (rows) and architecture-sensitive usability mechanisms (columns) that we have considered. A detailed description of this relationship is given in [18].

Table 1. Relationship between Usability Properties and Architecture-sensitive Usability Mechanisms

Usability Properties	Architecture-sensitive Usability Mechanisms											
	Different languages	Feedback	Undo	Form/Field validation	Wizard	User Profile	Cancel	History Logging	Command Aggregation	Action for multiple objects	Workflow Model	Provision of Views
Keeping the user informed		X										
Error management												
Error prevention		X		X	X		X		X		X	
Error correction			X				X	X				
Consistency												
Guidance		X			X							
Minimize cognitive load									X		X	
Explicit user control			X				X			X		X
Natural mapping												
Ease of navigation										X		
Accessibility	X											
Adaptability						X		X				X

It should be noted that the properties of Natural Mapping and Consistency cannot be arranged around specific architectural usability mechanisms. The reason is that these properties require the performance of different tasks and activities throughout the entire development process rather than the application of particular solutions at the architectural level. For example, the provision of natural mapping between the user tasks and the tasks to be implemented in the system calls for software requirements to be elicited during the analysis process bearing in mind this objective, and the whole system must be designed according to these requirements. The same goes for consistency, which involves different activities throughout the lengthy development process of the original or new versions of the system and among different functionalities of the same version.

4. Studying the Implications of Usability Mechanisms into Software Architecture

To analyze the architectural implications of the architecture-sensitive usability mechanisms presented in Table 1, we worked with different practitioners asking them to incorporate these mechanisms into their developments, once they had made the design for the system considering none of such mechanisms. Specifically, we worked on two small real applications developed by final-year Computing students, one real application developed by one of our Master students, and another real application developed by one of the industrial partners of the STATUS project. If the practitioners modified their designs to incorporate a specific mechanism, then the respective mechanism can be considered to be architecture sensitive.

The exact process followed to study the relationship between the usability mechanisms and the software architecture was:

- We worked with a list of usability mechanisms longer than the one that appears in Table 1, and compiled from HCI literature about specific software elements that improve system usability.
- We asked designers to build the design models for the systems without including usability mechanisms.
- We asked designers to modify their original developments to include the functionality for each of the mechanisms under consideration.
- If the modifications made affected the design models, for example, involved the inclusion of new components or different interactions between existing components, we considered that the mechanism was architecture sensitive and

generalized the design solutions provided by the different practitioners for these mechanisms.

- If the modification did not affect the design models (typically they affected in this case to lower level functions or pseudocode) then the mechanisms was considered non architectural sensitive.

An example of the architectural implications of one of the architecture-sensitive usability mechanisms (Undo) is shown in Figure 1. The complete demonstration of the architectural impact of the mechanisms shown in Table 1 appears in [19], including a detailed description of the design solutions provided for the practitioners for each mechanism, how they were derived, and an example of the inclusion of these mechanisms in a specific application. Note that the generalized architectural solutions for each mechanism (like the one shown in Figure 1) represents just one possible way of incorporating such usability mechanisms into a software design. Its goal is just show the architectural implication of a mechanisms but not at all the only solution to design such mechanisms.

5. Conclusions

Usability is a key issue in software development. This paper has shown an approach for dealing with usability from an architectural point of view. In particular, we have shown how usability has a real impact on software architecture, not only affecting the user interface as usually thought. Therefore, it is important to bear in mind the concept of usability when designing the overall system functionality and not just when designing the user interface.

The approach followed to illustrate the relationship between usability and software architecture focused on decomposing the concept of usability into lower levels that are progressively closer to the solution domain: usability attributes, properties and mechanisms. While usability attributes come from traditional HCI attributes, usability properties are taken from existing tips and heuristics that can be found in HCI literature. Finally, architecture-sensitive usability mechanisms represent specific usability issues to be incorporated into a software system and that have a demonstrated impact on software architecture.

By the time being, developers can use this work to consider usability mechanisms to incorporate into their systems during software architecture design. However, we are expanding this work to better serve developers. Specifically, we are developing what we have referred to as architecture-sensitive usability patterns which package both usability solutions and design solutions to mechanisms. In these patterns we customize architectural implications of each mechanism for specific architectural

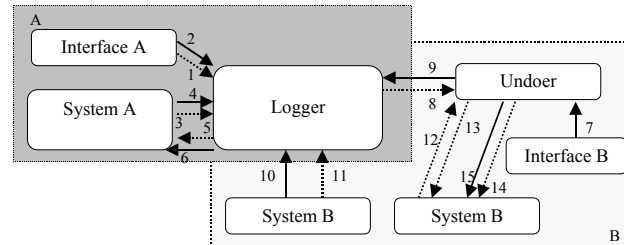
restrictions, for example, the use of MVC or PAC architectures; also we make explicit the user interface implications of these mechanisms to inform developers of what effect these mechanisms have on both the software architecture and the user interface.

So, although a lot of work still remains to be done to elucidate the exact details of the relationship between

software usability and software architecture, we have presented a first step that empirically demonstrate that there is such a relationship, and we have explicitly identified which usability issues involves such relationship.

- **Usability Mechanism:** The ability to undo an action and return to the previous state.
- **Example of design solution:**

- **Diagram:**



- **Participants:** This mechanisms design has two clearly separate parts. These parts have been labeled in the illustration as A and B, respectively. Part A collects the actions performed in the system (the number of actions to be stored will have to be specified when the system is developed) so that they can be later undone. Part B manages the respective undo.
 - **Interface A:** receives the request to execute an operation in the system, which may contain both the operation and data (1) (2). As we will see later, this execution request can also come from the actual system (3) (4).
 - **System A:** this module sends the functions and data executed in the system to the logger (3) (4) and also, optionally, if the logger does not store the actions internally, will send the information to the part of the system that manages these actions (5) (6).
 - **Logger:** this module receives the actions and the data requested by the user or from another part of the system (1) (2) (3) (4) and stores the logged action and data either internally or in another part of the system, in which case it will have to send this action and data to the system (5) (6) to be processed by the respective part of the system. Logger receives the undo request from Undoer (9) and, if the logged actions are stored in the logger, it then sends them one by one to Undoer (8). If they are not stored in the logger, it will receive both the data and the operation to be undone from another part of the system, which we have named System B, through (11) and (10), respectively.
 - **Interface B:** receives the undo request and sends it to Undoer through (7).
 - **Undoer:** sends the undo request to logger (9) and also sends each of the actions to be undone that it receives from logger to System B (13), as well as receiving the opposite operation to the one performed from System B (12). When it knows which opposite operation is to be performed, it sends the operation to System B along with the data associated with the operation in question through (14) and (15).
 - **System B:** it will search the system for both the action performed and the data associated with this operation (10) (11) if the data are not stored internally in the logger. It receives the actions to be undone (13) and provides the opposite operation (12) (for which purpose it will have to store what the opposite is for each action, see implementation section for example). The opposite action and the respective data will be sent to the respective part of the system ((15) and (14)).
- **Related mechanisms:** History logging is equivalent to part A of this mechanism. Therefore, if undo is provided, history logging could be provided at no extra cost.
- **Mechanisms implementation in OO:** This mechanism will generate an “undoer” class responsible for triggering the entire undo process. Additionally, there are the “listener” and “action-done” classes, which are used to store the actions that are performed as the system operates. A “system-action” class also has to be included to establish what the opposite is for each action that can be undone through the “is-the-opposite” relationship.
- **Example:** See [19] for a full example, not included here for reasons of space.

Figure 1. Architectural implications of the "Undo" mechanism

References

- [1] IEEE. *IEEE Std 1061: Standard for a Software Quality Metrics Methodology*. IEEE, 1998.
- [2] ISO. *ISO 9126-1 Software Engineering – product quality – part 1: Quality Model*. ISO, 2000
- [3] B. Boehm, J.R. Brown, H. Kaspar, M. Lipow, G.J. Macleod, M.J. Merritt. *Characteristics of Software Quality*. North Holland, 1978.
- [4] X. Ferré, N. Juristo, H. Windl, L. Constantine. “Usability Basics for Software Developers”. *IEEE Software*, vol 18 (11), p. 22-30.
- [5] L. Bass, P. Clements, R. Kazman. *Software Architectures in Practice*. Addison Wesley, Reading, MA, 1998.
- [6] L Bass, B. John, J Kates. *Achieving Usability Through Software Architecture*. Technical Report. CMU/SEI-2001-TR-005, March 2001.
- [7] J. Bosch and N. Juristo. “Designing Software Architectures for Usability”. *ICSE Tutorial*. Portland, OR, May 2003.
- [8] J. Nielsen. *Usability Engineering*. AP Professional, 1993.
- [9] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. Addison-Wesley, 1998.
- [10] J. Preece, Y. Rogers, H. Sharp, D. Benyon, S. Holland, T. Carey. *Human-Computer Interaction*. Addison Wesley, 1994.
- [11] L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, 1999.
- [12] D.L.Scapin, J.M.C. Bastien, *Ergonomic criteria for evaluation the ergonomic quality of interactive systems*, Behaviour & Information Technology, vol 16, no 4/5, pp. 220-231
- [13] B. Shackel. "Usability – context, framework, design and evaluation". In *Human Factors for Informatics Usability*. pp 21-38. Ed. by B. Shackel and S. Richardson. Cambridge University Press, 1991.
- [14] D. Hix, H.R. Hartson. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John Wiley and Sons, 1993.
- [15] M. Welie. The Amsterdam Collection <http://www.welie.com>, visited September 2003.
- [16] Tidwell. The Case for HCI Design Patterns. http://www.mit.edu/jdidwell/common_ground_onefile.htm, visited September 2003.
- [17] E Gamma, R Helm, R Johnson, J Glissades. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1998.
- [18] A. Andrés, J. Bosch, A. Charalampos, R. Chatley, X. Ferre, E. Folmer, N. Juristo, J. Magee, S. Menegos, A. Moreno. “Usability attributes affected by software architecture”. *Deliverable. 2. STATUS project*, June 2002. <Http://www.ls.fi.upm.es/status>
- [19] N. Juristo, A. Moreno, M Sánchez. “Techniques and Patterns for Architecture-Level Usability Improvements”. *Deliverable 3.4. STATUS project*. <Http://www.ls.fi.upm.es/status> May 2003.

Commonality and Requirements Analysis for Mesh Generating Software

Spencer Smith and Chien-Hsien Chen

Computing and Software Department, McMaster University

E-mail: smiths@mcmaster.ca

Abstract

This paper presents a proposal to improve the software quality of mesh generators, and by extension other scientific computation systems, by applying software engineering methodologies, in particular commonality and requirements analysis. The case study presented shows that mesh generating systems are well suited to development as a program family and that scientific computing problems need a requirements template tailored to this type of system.

1. Introduction

Software engineering methodologies have been gaining acceptance for many different types of software, such as business applications, real-time systems and safety critical systems. Unfortunately, scientific computing software has not yet received all of the benefits from the latest advances in software engineering. This paper will show that the development of scientific software, in particular mesh generating software, can greatly benefit from the use of such software engineering methodologies as commonality analysis and requirements analysis. Moreover, research in the field of software engineering will also benefit by tackling some of the unique challenges that arise during an analysis of scientific software. The case study presented in this paper illustrates some of these challenges and it provides an example of how to handle them.

Section 2 provides an overview of mesh generating systems, which provides the background for later discussion of the system. Section 3 addresses the question of why the software engineering methodologies of commonality and requirements analysis benefit mesh generating systems. Section 4 turns the discussion in the opposite direction and explains why mesh generators are different from other types of software and thus provide new and interesting challenges for software engineering researchers. The details of the commonality and requirements analysis for mesh generators, originally described in [6], are presented in Sections 5, 6 and 7. The final section consists of concluding remarks.

2. Mesh Generating Software

A mesh is a discretization of a geometric domain into small simple shapes, such as line segments in 1D, triangles or quadrilaterals in 2D, and tetrahedral or hexahedra in 3D. The principal application of interest for the current study is the finite element method, where meshes are essential in the numerical solution of partial differential equations arising in physical simulation [4]. The files created by a mesh generator must describe the following: how the domain is decomposed into cells; the material properties; and, the boundary conditions on the domain, with respect to applied tractions, prescribed displacements and fixity. The quality of the mesh that is generated is critical for the success of any finite element analysis; therefore, careful thought should occur before one proceeds to the implementation of the system.

3. Why is Predesign Analysis Necessary?

The predesign analysis that is advocated here consists of a commonality analysis and a requirements analysis. A commonality analysis is conducted to answer the question of whether the software should be designed as a program family. A program family is defined in [8] as a “set of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members.” The contributing factors may be the variation in application demands, the continuing improvement of technologies, the varieties of different algorithms, and so on. Instead of building those similar programs in the ignorance of their existence of one another, one should take the advantage of developing them as a family.

The analysis of why mesh generating software is well suited to development as a program family is postponed until Section 5. However, the idea has intuitive appeal when one considers the proliferation of mesh generating software. These software systems have much in common as they all produce mesh data, but they differ in the shape of the original domain, in the types of elements used, in the field of application of the resulting mesh, *etc.* Rather than have

many independent programmers working on very similar problems, it makes sense to investigate where the commonalities lie, extract that information, and then do the job once, do it well, and let future developers benefit from the expertise of their predecessors.

A commonality analysis can benefit both large and small mesh generating systems. In the case of large general purpose systems, they often have a long life, but they usually grow in an *ad hoc*. manner. Since the software grows without a plan, the data structures and algorithms that were well suited to what was initially a 2D code, may not be well suited to the eventual 3D code. Small mesh generators also benefit from a commonality analysis. In scientific computing it is common practise to develop a small specialized tool that is useful for only a restricted class of problems. A design based on the commonality analysis would allow the practitioner to pick out the parts that are specialized for their specific problem. Clear documentation of commonalities should lead to designs and implementations that are easier to reuse.

Software engineers generally advocate gathering and analyzing requirements in advance of building any software system because it is much easier and cheaper to correct mistakes and misconceptions at the beginning of the process than it is to try and fix problems during implementation and maintenance. During the process of requirement gathering, the requirements need to be documented in a software requirement specification (SRS), which includes the external behaviour of the system, the constraints placed on the implementation, the forethought about the life cycle of the system, and the acceptable response of the undesired events [7]. In terms of quality, a SRS should be correct, unambiguous, complete, consistent, modifiable, verifiable, and traceable [3]. In the case of mesh generating systems there is apparently no examples of SRS documentation. The absence of such documentation has the following consequences:

- Practitioners argue over the relative merits of different designs based on their own implicit requirements. A developer may criticize a design because it is not efficient, but this criticism would not be justified if the designer clearly started out with requirements that clearly stated that precision, maintainability and portability are more important than efficiency.
- Verification and validation (V & V) are difficult because it is unclear what standards the system is being verified and validated against? Clear requirements are necessary to know what the system should be inspected and tested for. Moreover, current V & V efforts focus on functional requirements, but it is the nonfunctional requirements, like accuracy, efficiency, portability *etc.*, that often distinguish designs.

- Communication between domain experts and others is difficult. Proper documentation will free domain experts from doing the implementation and allow them to pass this task onto computer science professionals. Moreover, it will ease communication between domain experts because they will be able to capture in the SRS the details and special cases that are not typically discussed in a journal paper.

4. Challenges for Predesign Analysis

The previous section explained how mesh generating systems can benefit from software engineering methodologies. This benefit actually goes in both directions, as research in software engineering can benefit by tackling the challenges presented by mesh generating systems. Some examples of these challenges include the following:

- Mesh generating systems are unlike the business applications that software engineers often focus on. For instance, mesh generators are not commercial products; they are often developed by researchers as open-source projects. Since mesh generators are different than business applications, the requirements template that is used should also be different.
- Software engineers often advocate the use of formal methods, which consist of using mathematical techniques and notations to specify qualities and attributes of the software. Mesh generators are based on the field of computational geometry, which also advocates the use of mathematics. The challenge is to bridge the gap of communication between these fields, but the exciting point is that both disciplines already speak in a common language, the language of mathematics.
- A goal that many software engineers hope to reach is the ability to proceed directly from specification to code in an automatic manner. In general this is an exceedingly difficult problem to solve, but if the domain of application is restricted, the problem may become feasible. The field of scientific computation is a good test bed for these kind of theories, because it consists of well-defined problems.
- Most software engineering methodologies rely on the use of discrete mathematics; therefore, scientific computing, including mesh generating, presents a challenge because the emphasis is on continuous mathematics.

5. Program Family Hypotheses

As indicated in [10], there are three basic assumptions underlying the production strategies of program families as

described below.

The Redevelopment Hypothesis

This hypothesis requires that most software development involved in producing the family should be redevelopment, which means that there should be a significant portion of the requirements, design, and codes in common among the family members. For mesh generators there are examples of large general purpose codes that are constantly redeveloped over their lifetime and most small codes are based on modification of existing code. Although different in the specific details, all mesh generators can be abstracted as: input information, calculate a mesh discretization and output the results.

The Oracle Hypothesis

This hypothesis requires that the types of changes that are likely to occur during the system's lifetime are predictable. This is certainly the case for a mesh generator as there is considerable literature on the topic and many example systems currently exist.

The Organizational Hypothesis

According to the organizational hypothesis, the program to be developed using the program family approach should be one that allows designers and developers to organize the software, as well as the development effort, in a way that the predicted changes can be made independently. If this assumption holds, then a predicted change will require changing only a few modules in the system. This hypothesis, however, is challenging for mesh generating systems.

For some of the likely changes, such as the changes in the user interface, visualization, and output format, the changes can be dealt with in an elegant way. However, for other types of the changes, like the use of different mesh generating algorithms and the use of different optimization and smoothing algorithms, the goal of restricting the change within one module is difficult to meet because a mesh data structure is inevitably assumed by these algorithms. Research on organizing the system so that predicted changes can be made independently has begun [5, 6].

6. The Commonality Analysis Document

The organization of the commonality analysis document basically follows the template proposed in [11], including the documentation of lists of commonalities, variabilities and parameters of variation. Each one of these lists assigns a unique item identification number to each entry, so that cross-referencing is possible. Also, the commonalities,

the variabilities, and the parameters of the variations are all stated in terms of external behaviour instead of the internal design and implementation, as stressed in [10].

The commonality section lists the assumptions that are true for all the members in the family. To make the list accessible, every commonality is organized into one of the following categories: problem domain, system capability, user input, system output, mesh information, timing and accuracy. The variability section contains all the changes expected in the capability of the family. The parameter of variation section further specifies variabilities by quantifying them.

An example commonality for mesh generators is, "A mesh should have only one element type throughout its entire domain." The related variability would be, "Different mesh generators may generate meshes of different element types." This variability could have the parameters of variation of "Triangles and Quadrilaterals."

7. Requirements Analysis

When designing the SRS, there are system-specific characteristics that should be taken into account so that the resulting SRS is better suited for the system it describes. The characteristics that are most important for an SRS for mesh generators are as follows: provision of the background concept/information is important, there are many rules and conventions, and the system features can be used to categorize the system's functionalities. These characteristics suggest the SRS structure that is discussed below.

7.1. Introduction

What should the introduction of the SRS discuss? In [1] and [9], the approach was taken to introduce the system right from the beginning, without mentioning the purpose of the documents. Whereas in [2] and [3], both the document and the system are introduced together in the first section. To keep the principle of separation of concerns, it was decided that the introduction on the system should be delayed to the section on the general system information, and the first section should only introduce the document.

7.2. General System Description

This section provides background knowledge about the system to help readers understand the specific system requirements. The information contained in this section is meant to be stated at high enough a level that other similar systems can borrow this section with only minor changes. In [9], the general information about the system is scattered in various different sections, which offers a less compact organization and violates the separation of concerns principle.

In [2] and [3], their overall system description sections have confusing and/or redundant subsections so that it is hard to distinguish between subsections in terms of their contents. Also, in many templates [1, 2, 3], major system constraints are discussed in this section. Constraints are one type of requirement, and can be addressed later in the system requirements section. As a result of the above discussion, the SRS for the mesh generator takes the skeleton of the general system description section from [1] and removes the subsection where constraints are discussed.

7.3. Specific System Requirements

After the general system information section comes the specific system requirements section. This is the major section of the SRS, and all system requirements should be here under their relevant subsection. There are three types of requirements: system constraints, functional requirement, and non-functional requirements. The SRS of the mesh generator takes some of its ideas from [2] and [3]. The functional requirements section has its template from [3].

7.4. Other System Issues

The idea of including supporting information relevant to the system development in a SRS is proposed in [9]. Examples of the materials included in this section are open issues, off-the-shelf solutions, project risks, project cost *etc.* The SRS of the mesh generator adopts this practise, but only chooses the subsections that are closely related to the process of developing the system. To be consistent with the program family methodology, there is a subsection named “Our Program Family”, which provides a blueprint of how the system will be extended.

8. Concluding Remarks

This paper presented a case study where software engineering methodologies are applied to a mesh generating system. The results of the commonality and requirements analyzes highlight how valuable these exercises are to mesh generating software, and to scientific computing systems in general. By identifying commonalities, much of the redundant effort that occurs today could be eliminated. Moreover, the introduction of a requirements template will help future research efforts in documenting scientific software. The usefulness of software engineering methodologies does not end at the predesign stages. A mesh generator can also benefit from methodologies for decomposing the system into modules and for documenting the interfaces of these modules [6]. This documentation allowed for a sample system to be implemented by individuals not involved in writing the original requirements document. The questions that they

raised acted as an effective review of the documentation and helped to improve its overall quality.

Acknowledgements

The financial support of the Natural Sciences and Engineering Research Council (NSERC) and of Material and Manufacturing Ontario (MMO) are gratefully acknowledged.

References

- [1] IEEE Std. 1233, *IEEE guide for developing system requirements specifications*, 1996.
- [2] ESA PSS-05-0 Issue 2, *ESA software engineering standards issue 2*, 1991.
- [3] IEEE Std. 830, *IEEE recommended practice for software requirements specifications*, 1998.
- [4] Marshall Bern and Paul Plassmann, *Mesh generation*, Handbook of Computational Geometry, Elsevier Science, 2000.
- [5] Guntram Berti and George Bader, *Design principles of reusable software components for the numerical solution of pde problems*, presented at the the 14th GAMM-Seminar Kiel on Concepts of Numerical Software (January 23rd to 25th, 1998).
- [6] Chien-Hsien Chen, *A software engineering approach to developing mesh generators*, Masters thesis, McMaster University, Hamilton, Ontario, Canada, 2003.
- [7] Kathryn L. Heninger, *Specifying software requirement for complex system: New techniques and their application*, IEEE Transactions on Software Engineering, vol. 6, no. 1, pp. 2-13 (January 1980).
- [8] David Parnas, *On the design and development of program families*, IEEE Transaction on Software Engineering, vol. 5, no. 2, pp. 1-9 (March 1976).
- [9] James & Suzanne Robertson, *Volere requirements specification template edition 6.1*, Atlantic Systems Guild, 2000.
- [10] David M. Weiss and M. Ardis, *Defining families: The commonality analysis*, Proceedings of the Nineteenth International Conference on Software Engineering, pp. 649-650 (1997).
- [11] David M. Weiss, *Commonality analysis: A systematic process for defining families*, ESPRIT ARES Workshop '98, pp. 214-222 (February 1998).

Contextual Comparison of Discovered Knowledge Patterns

A.G. Büchner*, M. Baumgarten*, J.G. Hughes*, W.D. Patterson*

* Northern Ireland Knowledge Engineering Laboratory
Faculty of Engineering, University of Ulster
{ag.buchner, m.baumgarten, jg.hughes, wd.patterson}@ulster.ac.uk

Abstract: *Contextual comparison of discovered patterns deals with the interpretation of outputs from data mining algorithms. The vehicle provided to perform such operations is that of contextual interestingness, which allows the allocation of importance and direction to each attribute in a result set. Applying these mechanisms it is not only possible to detect trends in results across time, but also to compare individual result elements.*

Keywords: data mining, knowledge comparison, interestingness, context

1 Introduction

Recently, dissemination, application and deployment of results, which have been generated by knowledge discovery, have been of interest to the research community. The main objective is that once useful and novel information has been discovered it can be utilized in a given domain. A problem which occurs frequently in commercial and research scenarios, is that results from knowledge discovery exercises carried out in separate contexts have to be compared. The objective of this paper is to address this issue.

For example, a classification is carried out on the customer base of a retail outlet to discover distinguishing behavior between customers with and without loyalty cards. After the introduction of special offers targeted at loyal customers, another analysis is carried out. The question one would ask subsequently is “How much better are the results based on the introduced campaign?” However, no mechanism exists at present to carry out such a comparison operation. Similarly, when evaluating different algorithms for the discovery of the same type of patterns, the results cannot be compared without mundanely stepping through the details of each result set.

Section 2 defines the problem and outlines the scope of the paper. In Section 3, a generic interestingness framework is presented and notational issues are addressed. Section 4 applies the proposed framework before an application in the web mining arena is presented in Section 5. Section 6 concludes the paper.

2 Problem Definition

Results from knowledge discovery stem from different contexts. The types of context which are relevant are *algorithm contexts* (same data is applied to algorithms, e.g. ID3 and C5), *data contexts* (different data / same algorithm and threshold settings, e.g. from different time spans or samples), *parameter contexts* (parameters such as

thresholds are modified using the same algorithm and data), or any permutation thereof. Additionally, analysts interpret results from different viewpoints (*user context*).

In order to allow the interpretation of results generated in such disparate situations, it is necessary to have access to a flexible, yet powerful, mechanism which allows the comparison of knowledge. Issues which arise are

- What types of knowledge can be compared with each other?
- How can contextual information be incorporated?
- What is the most appropriate equivalence mechanism to be applied in order to perform comparisons?

This work will resolve these issues and provide a novel contextual interestingness measure which can be used for comparison of results from knowledge discovery.

The high-level calculation of the contextual interestingness calculation θ of any knowledge component of type in a certain *context* is formulated as follows.

$$\theta_{\text{Type}}^{\text{Context}} \rightarrow [0..1] \quad (1)$$

The greater the result the more interesting it is. The objective is to specify this calculation with the greatest degree of flexibility and support of contextuality.

Silberschatz & Tuzhilin [1] have tackled the central problem of ‘good’ measures to identify the interestingness of a pattern by introducing two different kinds of interestingness. Objective measurements relate to the structure of a pattern object and the underlying data used to discover them, while subjective measurements depend on the user’s needs, the domain the data is analyzed in, and the scenario to which they are applied. While their approach allows the comparison of interestingness values, it neither provides a vehicle to allow a user to define the concept of comparative or contextual interestingness. Further work related to the proposed approach covered by the three areas of *knowledge fusion* which deals with the combination of knowledge, *knowledge sharing* which refers to the process of locating and extracting knowledge from multiple sources and transforming it so that the union can be applied in problem-solving and *sequence alignment methods* which calculate the distance between sequences, which is reflected by the numbers necessary to convert a source sequence into its target counterpart.

3 Interestingness Framework

This section introduces a framework which provides structures and operations for the comparison of multiple

results from knowledge discovery. The principle idea is not to compare knowledge per se, but to compare the results which are derived from discovered knowledge.

3.1 Result Comparison Structure

The outcome of a knowledge discovery exercise is a predictive model. Each model is of a certain type, for instance a neural network or a set of sequences. Models of some type contain also the information about the data it has been derived from (associations, sequences, episodes), while most types only provide information about the model itself (rules, clusters, neural nets, regression, etc).

While it is in principle possible to compare results of different type e.g., a neural net with a decision tree, the scope of this work is restricted to the comparison of compatible results, i.e. results of the same type. All results of the same type t are organized in a result space \mathcal{R} .

Definition 1. Result Space \mathcal{R}

$$\mathcal{R} = \{R_1, R_2, R_3, \dots\}, \text{ such that } \left| \bigcup_{i=1}^{|\mathcal{R}|} t(R_i) \right| = 1. \quad \blacklozenge$$

Each $R \in \mathcal{R}$ contains a set of result elements which describe the result with quantitative and / or qualitative values. Quantitative measures represent information about the result element per se e.g. support and confidence. Qualitative measures provide information about the content of a result element e.g. quantification of values.

Definition 2. Result R and Result Elements r

$R = \langle \{r_1, r_2, r_3, \dots\}, a \rangle$; each result element $r = \langle I, a \rangle$, where I is an optional set of items and a a set of attribute tuples such that $a = \{ \langle \lambda_1, v_1 \rangle, \langle \lambda_2, v_2 \rangle, \dots \}$, where λ represents a label and v its normalized value ($0 \leq v \leq 1$). \blacklozenge

Quantitative and qualitative values are treated holistically and are referred to as attributes. A set of attributes can be attached to each element in \mathcal{R} and R . The range of available attributes depends on the type of knowledge that has been generated. A topology containing results and allotted attributes is depicted in Figure 1.

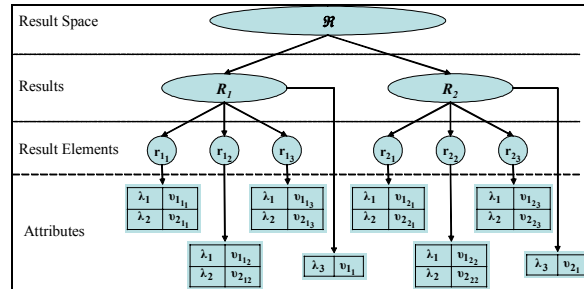


Figure 1. Result Space Topology

In order to illustrate the outlined concepts an example is provided dealing with the results of a decision tree. The objective of the exercise is to apply a model on two data sets (1998 and 2003) of the same patients, which predicts the likelihood of developing Alzheimer's disease. The

input data contains information on patients' gender, age, height, weight, smoking habits, alcohol consumption, etc.

A rule induction example has been applied with three classification labels (low, medium and high) for the predictability of the disease, which are represented as items. The output sets contain two attributes, namely support and confidence.

R_i	I_{1998}	$a_{support}$	$a_{confidence}$	R_i	I_{2003}	$a_{support}$	$a_{confidence}$
r_{11}	Medium	0.04	0.4	r_{21}	High	0.05	0.5
r_{12}	High	0.05	0.2	r_{22}	High	0.02	0.8
r_{13}	Low	0.07	0.8	r_{23}	Medium	0.07	0.6
r_{14}	Medium	0.01	0.4	r_{24}	High	0.02	0.2

Table 1. Rule Induction Results

The sample result elements are shown in the table above. Questions that are feasible to ask are: "Has the health of patients improved or deteriorated?" and "Which patient's condition has improved / deteriorated over the last 5 years?"

3.2 Contextual Interestingness

In order to compare attributes and results, the notion of contextual interestingness is introduced on attribute, element and result level. In order to allow the user to specify contextuality for a given problem, the concept of contexts is introduced. A context describes a given phenomenon, scenario or problem, using knowledge discovery specific attributes. Contexts are organized in a context space Γ .

Definition 3. Contexts

$\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \dots\}$, where each $\gamma = \{\alpha_1, \alpha_2, \alpha_3, \dots\}$. Each attribute $\alpha = (\lambda, \delta, i)$, where $\delta \in \{0, 1\}$ and $0 \leq i \leq 1$. \blacklozenge

The label λ is the name of a phenomenon in a given problem space, aka context identifier. Examples in data mining are thresholds (support, confidence) or quantitative information (weight, size, and length). The label is the logical link between result attributes and context attributes. The direction δ is a binary value that states whether an increase of the value is positive (0) or negative (1) in the context the data mining exercise is carried out. The importance factor i states the relevance of the direction δ .

For example, when discovering sequences in a web mining application, long sequences are attractive when the host's remuneration is based on the number of page impressions. Contrarily, short sequences are appealing if the objective is that customers solve their problem with as few clicks as possible. Both importance and direction are adjustable within the given limits.

The interestingness of an attribute comprises the degree of interest associated with it in a given context. That is when putting a certain result in a certain context, its interestingness can be calculated.

Definition 4. Attribute interestingness θ_a

$$\theta_a(v, \delta, i) = |(v - \delta) * i| \rightarrow [0 \dots 1] \quad \blacklozenge$$

Interestingness embraces both objective and subjective measurements. Former relates to the structure of a pattern and the underlying data used to discover it, while latter

depends on the user's needs, the domain the data analyzed is in, and the scenario to which it is applied. Thus, the interestingness of a pattern is by no means an objective value that remains constant across comparisons; interestingness is a subjective representation of the user's priorities in conjunction with the raw pattern values.

In order to compute the interestingness θ_o of the attributes of either a result element r or a result R all attributes are taken into account.

Definition 5. Object interestingness θ_o

$$\theta_o(a) = \left(\sum_{i=1}^{|a|} \theta_a(v_i, \delta_i, t_i) \right) / |a| \rightarrow [0..1] \quad \blacklozenge$$

$|a|$ represents the amount of attributes in a . This operation calculates the arithmetic mean of all attribute values in a given context. This measure is used as basis for the calculation of the result element interestingness θ_e and the result interestingness θ_r .

Definition 6. Element interestingness θ_e

$$\theta_e(r) = \theta(a(r)) \rightarrow [0..1] \quad \blacklozenge$$

Definition 7. Result interestingness θ_r

$$\theta_r(R) = \theta(a(R)) \rightarrow [0..1] \quad \blacklozenge$$

Using the earlier introduced Alzheimer's prediction example, the classification labels have been quantified to low=0.3, medium=0.6 and high=1. The following can be calculated, given that for $\lambda_{\text{risk}} \delta=0$ (in this context lower is more interesting), $t=100\%$ (highest importance), for $\lambda_{\text{support}} \delta=1$, $t=50\%$ and for $\lambda_{\text{confidence}} \delta=1$, $t=80\%$.

$$\begin{aligned} \theta_e(r_{11}) &= \frac{|(0.6-1)*1.0| + |0.04*0.5| + |0.4*0.8|}{3} = 0.247 \\ \theta_e(r_{21}) &= \frac{0 + |0.05*0.5| + |0.5*0.8|}{3} = 0.142 \end{aligned} \quad (2)$$

Due to the fact that interestingness of the result in 1998 is greater than the one of 2003, this means that patient r_1 's condition has worsened. Calculating the four result element interestingness measures for 1998 and 2003, and building the arithmetic mean results to 0.252 and 0.18, respectively. Those two values are accepted as new attributes λ_{risk} for R_1 and R_2 ($\delta=0$ and $t=100\%$). Given that Alzheimer's disease is an age related illness the data set of 2003 is 'punished' via an age attribute ($\delta=1$, $t=20\%$).

R	$\left(\sum_{i=1}^4 \theta_e(r_i) \right) / 4$	δ_1	t_1	age	δ_2	t_2
1998	0.252	0	100%	0.5	1	20%
2003	0.18	0	100%	0.8	1	20%

Table 2. Result Attributes

Calculating $\theta_r(R_1)=0.176$ and $\theta_r(R_2)=0.38$ it can be shown that the overall population has also deteriorated; in fact the situation has worsened substantially.

3.3 Attribute Generation

As outlined previously, the set of attributes a associated

with each r and R of the result space form the basis of calculating contextual interestingnesses. Attributes on each level can be provided through the result sets themselves, they can be specifically set by the user or they can be derived from other attributes, e.g. average or coverage values. Average values represent the arithmetic mean of sub values, that is the average interestingness of all $r \in R$ calculated as follows.

$$\left(\sum_{i=1}^{|R|} \theta_e(r_i(R)) \right) / |R| \quad (3)$$

Coverage values are derived using the scope of the overall result space. For instance, given all distinct result elements of r , a coverage attribute for each R is calculated indicating the exposure of r in each R .

4 Specific Knowledge Types

The proposed framework has been applied to results from a rule induction exercise. In order to show that the introduced mechanisms can be adapted to every common type of knowledge discovered by a predictive modeling exercise, the specific knowledge type of sequences is presented in detail, before other types are briefly covered.

In addition to the generic structural and operational artifacts presented in Section 3, a collection of standard segment-specific attributes is introduced. A pictorial summary of the constructs is depicted in Figure 2 below. Typical standard attributes on result element level are segment density, size and weight. More domain (or context) specific attributes can be added to this set, for instance average price per segment. Attributes on result level include the number of segments in all r 's or average segment size, density and weight. The inclusion of the mean element interestingness of all r 's has proven useful (see example above).

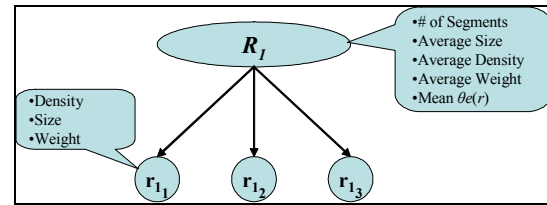


Figure 2. Segment-specific Structure

It is possible to introduce attributes at result space level, for example, total number of segments, total number of unique segments or average segment size, density and weight. These attributes have proven useful when calculating the interestingness of \mathcal{R} , for instance, to determine the quality of the entire result space. However, due to the fact that they cannot be used for comparison purposes (only one knowledge space exists for each type), they were not included in this work.

Due to space restrictions it is not possible to cover a wide range of knowledge types in detail. The table below lists a range of attributes for a list of important knowledge types.

Type	Element	Result
Associations	# of items	# associations
	Support	Avg. / unique # items
	Confidence	Support / confidence
Sequences	# transactions / sets	# sets
	Max. time	Occurrence
	Max. support	Support / Confidence
Decision Trees	# nodes	Score
	# scores	Record count
	Avg. predicate number	# predicates
Neural Networks	# neurons	Avg. bias / threshold
	Avg. bias	Avg. connection weight
Regression	Avg. # predictors	# predictors
	Max. intercept	Intercept
	Avg. coefficient	Avg. coefficient
Naïve Bayes	Threshold	# pair counts
	# Bayed input	Avg. value / count
	Avg. value / counts	Max. value

Table 3. Knowledge Type Attributes

It must be stressed that the set of covered attributes and methods is by no means complete and can be extended at any time. For instance, in the application described in Section 5, an additional ‘coverage’ measure has been introduced, which conveys the number of items in a rule related to the number of items in the respective result.

5 Application

The presented domain-agnostic interestingness framework has been integrated in a knowledge comparison architecture, where a web mining application has been conducted. The goal was to show the changes of visitor clusters in two time frames (contexts). After the analysis of the first period, changes were carried out to the site and the objective was to demonstrate the impact of those changes. A representative investor relations segment from each time frame is used to show the application of the framework.

During the first period the cluster contained eight pages, which are listed in Table 4a in conjunction with the rounded average number of seconds spent on each page.

Page	Seconds spent	Page	Seconds spent
Home	5	Home	5
Staff/CEO	22	Seniorstaff	33
Staff/CIO	20	Investment	48
Staff/CFO	20	Endorsements	11
Investment	45	Board	9
Board	16		
Awards	8		
Endorsements	8		

Table 4. Cluster from (a)Context₁ and (b)Context₂

The three staff pages were amalgamated to a senior staff page and the content of the awards page was included in an endorsement page (Table 4b). The following table shows the attributes and its values for both contexts and their respective importance factors and directions.

Attribute	r ₁	r ₂	ι	δ
λ _{AllPages}	8 (1.0)	5 (0.625)	100%	↓
λ _{TimeSpent}	101 (0.73)	139 (1.00)	100%	↑
λ _{Density}	0.42	0.37	75%	↓
λ _{Size}	84 (0.74)	113 (1.00)	10%	↑
λ _{Weight}	0.17	0.22	75%	↑

Table 5. Attributes for r₁ and r₂

Following the element interestingness θ_e calculation in Definition 6, the comparison of the two segments representing investor relations, was calculated as follows.

$$\theta_e(r_1) = \frac{0 + 0.73 + 0.435 + 0.074 + 0.623}{5} = 0.372$$

$$\theta_e(r_2) = \frac{0.375 + 1 + 0.473 + 0.1 + 0.585}{5} = 0.5065$$
(5)

The cluster from the second time frame (r₂) is significantly greater (that is, better within the scope of the analysis) than the first ($\theta_e(r_1) < \theta_e(r_2)$). Thus, it was possible to measure the impact to the investor segment of the changes which were made to the web site structure.

A further comparison was carried out which compared the two clusters in their entirety (all results of both contexts). The result interestingness θ_r was performed according to Definition 7, based on the derived avg. element interestingness of all r's, the number of clusters, the avg. time spent, the avg segment size and weight.

Attribute	R ₁	R ₂	ι	δ
λ _{Avgθ_e}	0.44	0.58	100%	↑
λ _{AllCluster}	6 (0.75)	8 (1.00)	0%	↑
λ _{AvgTimeSpent}	34 (0.89)	38 (1.00)	100%	↑
λ _{AvgSize}	417 (1.00)	364 (0.87)	50%	↑
λ _{AvgWeight}	0.25	0.22	75%	↑

Table 6. Attributes for R₁ and R₂

Calculating the result interestingness for R₁ and R₂ results in $\theta_e(R_1)=0.404$ and $\theta_e(R_2)=0.436$, which shows that the new site is more interesting. However, the change is not as significant as the one for the investor relationship segment, which was the intention of the restructuring.

If the target of the analysis would be the improvement of the click-to-close ratio, the direction of $\lambda_{AvgTimeSpent}$ would be reversed. Keeping all other values static results in $\theta_e(R_1)=0.248$ and $\theta_e(R_2)=0.236$. This hypothetically example demonstrates the simplicity and flexibility of the framework and shows how the same results of knowledge discovery can be analyzed in multiple contexts.

6 Conclusions

A generic framework has been presented that allows the comparison of results, which are discovered by knowledge discovery. The framework is algorithm agnostic, that is it covers all common types of knowledge (*generality*). Due to its flexible structure, full *extensibility* and *re-usability* are guaranteed. The vanilla approach of the model and its calculations assure *simplicity* and *integratability*. All contextual values can be adjusted interactively (*flexibility*) providing a solid domain-agnostic basis (*applicability*).

References

- [1] A. Silberschatz, A. Tuzhilin, What Makes Patterns Interesting in Knowledge Discovery Systems, in *IEEE TKDE*, 8:970–974, 1996.

Datawarehouses design: effectivity of the star schema

Coral Calero, Manuel Serrano, Mario Piattini

ALARCOS Research Group

Department of Computer Science-University of Castilla-La Mancha

Paseo de la Universidad, 4 13071 Ciudad Real (Spain)

{Coral.Calero, Manuel.Serrano, Mario.Piattini}@uclm.es

Abstract. Datawarehouses have become the most important trend in business, and it is essential that the design be made to assure efficiency and simplicity of use. Although it is generally accepted that the star design is the best way to implement datawarehouses in relational database management systems there are no studies that confirm this assumption. With the aim of determining if the star design really gives more comprehensible datawarehouses, we are carrying out a series of experiments. In this article we present the studies done up to now showing that there is no difference in difficulty when using the “traditional” relational method (using E/R modeling and then transforming it into the relational model) than when using the star design.

1. Introduction

Datawarehouses arose due to the need of organizations to have mechanisms that helped in decision making. Datawarehouses have become the most important trend in organizations since they provide information for improving strategic decisions. Datawarehouses and related business intelligence technologies have increased enormously in recent years, and are expected to reach 150 billion dollars in 2005 (Jarke et al., 2000; Chenoweth et al., 2003).

If the datawarehouse has been properly constructed, it provides the organization with a foundation that is extremely flexible and reusable (Inmon, 2002). So, it is essential to assure that the datawarehouse is designed properly. One way of designing a datawarehouse is to use dimensional modelling, a logical design technique alternative to the classical database design based on entity-relationship modelling (together with the transformation to the relational model).

The dimensional modelling technique seeks to present the data in an intuitive standard framework that allows high-performance access. The dimensional model is composed of a table called the fact table (the primary table that is

meant to contain measurements of the business) and a set of smaller tables called dimensional tables.

Some authors, such as Kimball et al (1998) argue that dimensional modelling is the only viable technique for delivering data to end users in a datawarehouse and has a number of advantages over the entity-relationship based methodology. In general, it is argued that decision-oriented dimensional datawarehouses are fundamentally different from transaction-oriented relational databases and a different set of tools is required for their effective development (Chenoweth et al., 2003).

There is, however, no proof of this assumption. In an attempt to clarify this assumption, we decided to do a series of experiments with the intention demonstrating that the use of the star model makes datawarehouses simpler to use than when the traditional design is used (relational and E/R).

The work done until now can be divided into two studies (one of them was replicated twice). In this article we present all the results that we have obtained from all these experimental studies.

In the following section all the experimental processes of each of the studies together with its conclusions are described. Section three presents the conclusions and describes future studies.

2. Experimental work

The objective of our work is to determine if it is better to design the datawarehouse using a traditional design methodology (that begins by model E/R and follows with the transformation to the relational model) or by using the star design. Until now we have carried out two experiments and we have replicated one of them twice.

In both experiments the working hypotheses, the dependent and independent variables with which the experiments work, the experimental material, the execution process and many of the threats to the validity of the experiments are the same. Before explaining in depth each experiment and the results obtained we will explain in detail the common elements.

2.1. Hypotheses.

Our hypotheses are:

Null hypothesis (H0): There is no difference in the understandability of the two kinds of schema (traditional and star).

Alternative hypothesis (H1): There is a difference in the understandability of the two kinds of schema (traditional and star).

2.2. Dependent and independent variables.

In every case we have measured the dependent variable (understandability) by means of the time used by each subject in making the indicated tasks, the independent variable being the model used for the logical representation of the datawarehouse (traditional or star).

2.3. Experimental material and execution

The experimental material and the form in which the experiments were carried out were similar in both experiments. The schematas were provided to the subjects (six in the first study and twelve in the second) together with the questions. The subjects must indicate how and to what tables of each schema they must gain access to recover certain information from the datawarehouse. They also had to write down the time used in responding to each of the questions.

The experiment was done in one session. Before carrying out the experiment we gave an intensive explanation of what kind of problems they had to solve, how to answer the questions and what sort of material was provided for the accomplishment of the experiment. In no case did the subjects have knowledge of the aspects that we were trying to study or of the hypotheses we were working with.

2.4. Threats to validity

In order to be able to avoid diverse threats to the validity of the experiment, we tried to control some aspects:

- In each experiment subjects had similar experience and knowledge. Although to work with students might not appear rigorous, there exist studies that affirm that the differences between students and professionals are small and studies with students are viable under certain conditions (Hörst et al., 2000).
- The domains of the diagrams were simple and sufficiently known to avoid problems of understanding.
- In order to avoid learning effects the schematas were given to each subject in a different order.

- As it was the first time that the subjects performed an experiment of this type, persistence effects do not exist.
- Subjects were motivated because the exercises were included into the knowledge they had to acquire in their training.
- Plagiarism was controlled and conversations were not allowed among subjects during the experiment.
- All the doubts were solved by the person who led the experiment.

In the following two sections we will present in depth individual aspects of each of the experimental studies.

3. First experiment

As we have already mentioned this first experiment was replicated twice so, finally we had three examples of the same experiment (an original experiment and two replicas).

3.1. Experimental design

The experiment consisted of six schematas, three traditional schemes and three semantically equivalent schemas designed using star diagrams. Subjects had to formulate SQL queries and to write down the time (in seconds) that it took make them.

3.2. Subjects

In the original experiment the subjects were 11 PhD students at the School of Computer Science of the University of Castilla-La Mancha (UCLM) in Spain. One of the replicas was made by 12 undergraduate students of the UCLM who were enrolled in the final year (when the experiment was done they were following a course on databases lasting two semesters) whereas the other replica was carried out with 24 PhD students of the Pinar del Rio University (Cuba).

In all the cases the subjects had knowledge of design and use of databases and datawarehouses. In addition the PhD students had a deeper knowledge of datawarehouses because they had received this information as part of their PhD studies.

3.3. Limitations

We are conscious of some limitations associated with these experiments such as the small number of subjects and objects and the difficulty associated with the use of SQL for the specification of the exercises given to the subjects.

3.4. Results

For our experiment we fixed a value $\alpha = 0.1$ to increase the power of the statistical tests (that is to say, the probability of rejecting our hypotheses when these are false). Due to the experimental design and to the gathered data the most suitable test is a repeated measure univariate ANOVA test (SPSS 11, 2001).

In tables 1, 2 and 3 the results obtained after applying the statistical test to the data gathered from the experiment are shown. Analyzing the significance value we can observe that all the values are greater than α and, therefore, we cannot reject the null hypothesis. So, there is no difference in the time used to answer the questions depending on the type of design used (Schema_type variable).

Table 1. ANOVA results for PhD students from UCLM (Spain)

	Sum of squares	Df	Mean Square	F	Sig.	Power
Schema_Type	10730,09	1	10730,09	0,01	0,95	0,10

Table 2. ANOVA results for undergraduate students from UCLM (Spain)

Source	Sum of squares	Df	Mean Square	F	Sig.	Power
Schema_Type	8557,55	1	8557,55	0,09	0,82	0,10

Table 3. ANOVA results for PhD students from Pinar del Rio University (Cuba)

Source	Sum of squares	Df	Mean Square	F	Sig.	Power
Schema_Type	101117,21	1	101117,21	0,02	0,90	0,10

As a conclusion we can deduce that there seems to be no difference in the understandability of the schematas because of the design method used. Nevertheless, as these conclusions could be due to the sizes of the schematas used in the experiment (they are not very large) or to the fact that SQL has been used to obtain the answers, we decided to replicate the experiment incorporating some changes that allowed us to avoid these limitations as far as possible.

4. Second experiment

In the second experiment and taking into account the previously obtained results, we decided to make a replica in which, working with the same hypotheses and with the same variables, we incorporated two fundamental changes; firstly we decided to increase the number of objects and, secondly, we tried to facilitate the work of

the subjects allowing them to use natural language instead of the SQL.

Thus, the new experiment can be characterised as follows.

4.1. Experimental design

In this occasion, the experiment consisted of twelve schemes (the six ones from the first experiment and other six new schemas), six traditional schematas and six semantically equivalent schematas designed with star diagrams. On each one of them, the subjects had to indicate (in natural language), the necessary steps to obtain certain data from the datawarehouse schema and to write down the time (in seconds) that they took to perform these steps (as in the previous case, since, as we have already indicated, the dependent variable did not vary).

4.2. Subjects

In this occasion we had eighteen people, final year undergraduate students of the School of Computer Science of Ciudad Real (UCLM) who were doing a course on Information Retrieval where all the concepts related to datawarehouses were explained. In addition, all of them had attended a course on Databases (mandatory at the third level of their studies) in which all the contents relative to the relational model are dealt with in depth.

4.3. Limitations

The main limitation in this case is the low number of subjects.

4.4. Results

We fixed a value $\alpha = 0.1$ and, as the design is the same as in the previous experiment, we applied also the repeated measure univariate ANOVA test (SPSS 11, 2001).

In table 4 the results obtained from the experiment are shown. As the value obtained is less than α , we can reject the null hypothesis, and therefore there is a difference in the time needed to answer the questions according to the type of design used (traditional or star, Schema_type variable).

Table 4. ANOVA results for the second experimental work

Source	Sum of squares	Df	Mean Square	F	Sig.	Power
Schema_Type	154615,005	1	154615,005	11,572	0,001	0,960

As the obtained value is significant, the next step is to take the Difference of means to obtain more data. In table 5 the results obtained for this statistical test appear.

Table 5. Results of the Difference of means for the second experimental work

Number of Schema	Traditional design	Star design	Difference of means
1	174,33	164,56	9,78
2	263,78	259,72	4,06
3	245,22	243,61	1,61
4	333,83	152,67	181,17
5	168,00	158,61	9,39
6	299,83	184,78	115,06
Total	247,50	193,99	53,51

Based on the results obtained for the Difference of means, it can be concluded that when the datawarehouse is designed using star diagrams the time averages are smaller than when we use traditional design, and so we could conclude that the star design seems to be easier to understand than the traditional one for the design of datawarehouses.

5. Conclusions from all the experimental work developed

As can be appreciated, the results obtained are not definitive. Although in the first study we obtained the result that both modelling techniques could be appropriate for the logical design of the datawarehouses in the second study the star model seems to be more suited.

So, it seems that, at least, the use of the star design is not more difficult than the traditional design.

Nevertheless, to be able to reach more definitive and trustworthy results it is essential to carry out replicas of the second experimental work with more subjects with differing experience (for example with professional designers of datawarehouses). In addition, it would be advisable to perform another type of replica, for example, varying the hypotheses.

6. Conclusions and future works

Datawarehouses are one of the main trends in information systems since they help in strategic decision making. Diverse methods for datawarehouse design have been set out based on star diagrams, since it is supposed that these diagrams, compared with the use of traditional modeling

(ER and relational), increase the effectiveness and the understanding of the schemes of datawarehouses.

Although this affirmation is widely accepted, it has not been empirically demonstrated, which is why we decided to do a series of experiments. The experiments try to detect causal relationships between the logical design of a datawarehouse (traditional vs star) and the understandability of it. In all of them subjects had to make queries (in SQL or natural language) about a logical datawarehouse schema (traditional or star design). The way to determine the understandability of each of the schematas was to record the time required to carry out the indicated operations.

As a conclusion of our study we can say that using the star model, as was anticipated, we obtain schematas not more difficult to understand than the relational ones and, in some cases, they have turned out to be simpler.

To be able to have reliable results and definitive conclusions it is necessary to perform more replicas of these experiments and also new experiments together with their replicas. In addition, it is also fundamental to run case studies to know if the results obtained from controlled experiments are the same.

ACKNOWLEDGMENT

This research is part of the MESSENGER project (PCI-03-001) supported by the Consejería de Ciencia y Tecnología of Junta de Comunidades de Castilla-La Mancha (Spain) and the CALIPO project (TIC 2003-07804-C05-03) supported by the Ministerio de Ciencia y Tecnología (Spain).

REFERENCES

- [1] Hörst, M., B. Regnell and C. Wohlin (2000) Using students as Subjects – A Comparative Study of Students & Professionals in Lead-Time Impact Assessment. 4th Conference on Empirical Assessment & Evaluation in Software Engineering, EASE, Keele University, UK.
- [2] Chenoweth, T., Schuff, D. and St.Louis, R. (2003). A method for developing dimensional data marts. Communications of the ACM. December 2003. Vol. 46, No.12. pp. 93-98
- [3] Inmon, W.H. (2002) Building the data warehouse. 3rd ed. Ed. Wiley
- [4] Jarke, M., Lenzerini, M., Vassilou, Y. and Vassiliadis, P. (2000) Fundamentals of Data Warehouses. Ed. Springer.
- [5] Kimball, R., Reeves, L., Ross, M. and Thornthwaite, W. (1998) The datawarehouse life cycle toolkit. Ed. Wiley
- [6] SPSS 11.0. (2001) *Syntax Reference Guide*,. Chicago, SPSS Inc., 2001.

Distributed Knowledge Based System Using Grid Computing for Real Time Air Traffic Synchronization - ATFMGC

Li Weigang¹, Daniel Amaral Cardoso², Marcos Vinícius Pinheiro Dib², Alba Cristina Magalhães Alves de Melo¹

Department of Computer Science of the University of Brasilia¹

Politec Brazil²

email: weigang@unb.br

Abstract. A distributed Knowledge-Based System using Grid Computing for Real Time Traffic Synchronization is proposed in this research. The paper presents Air Traffic Flow Management (ATFM) problem and its synchronization property. Air Traffic Flow Management Grid Computing-ATFMGC, the grid architecture, the basic components and relationships among them and the Knowledge-Based System (KBS) inference process are described in order to demonstrate the developed model. As an illustrative example, a tactical planning case study in São Paulo airport is reported. Using grid computing with Distributed Knowledge-Based System to improve ATFM computational efficiency may be a promise topic in the further research.

1. Introduction

Air Traffic Flow Management (ATFM) is a kind of Real Time Traffic Synchronization problem [1,2]. Due to the large-scale, safety and synchronization characteristics, in most of the models and systems for ATFM, computing efficiency is a common critic problem. Since 1970s, scientists from Artificial Intelligence (AI), Operation Research and Air Transportation have worked together to develop more efficient Air Traffic Control (ATC) and Air Traffic Management (ATM) systems, but the computing-based solution still needs to be further enhanced to reduce the aircraft delay.

Some Knowledge-Based Systems (KBS) have been developed in ATC/ATM, such as 4D-Planner that is a ground based planning system using a rule-based system for arrival sequencing and scheduling in ATC [3,4]. Gosling [5] pointed out the potential of AI application in ATC. A real time knowledge based support for ATM has been developed by IBM Switzerland [6]. In Brazil, an expert system for Air Traffic Flow Management (ATFM) has also been investigated to make timetable schedule and the traffic flow control [7,8].

A distributed ATM system was studied in Australia [9]. The advantages of that approach are inherent, autonomy, communication and reliability. Prevôt from NASA Ames Research Center has studied a distributed approach for operator interfaces and intelligent flight guidance, management and decision support [10]. An application of multi-agent coordination techniques in ATM, which sets up a methodological framework using multi-agent coordination techniques that supports the collaborative work in ATM has also presented recently by Eurocontrol [11]. It should be mentioned that, the multi-agent coordination techniques is a useful methodological framework, however the research in [11] is limited to a software shell. Due to the great quantity of traffic as ATFM, this implementation may be difficult when brought into practical fields.

Recently, grid computing presents a perspective to get the solution for the large-scale computation task as ATFM. Computational grid has been defined as “coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations” [13-17]. They consist of hardware and software infrastructure which provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities. An agent-based resource management system for grid computing shows the importance of the research in both grid computing and AI [12]. Grid resource management has been defined as the process of identifying requirements, matching resources to applications, allocating those re-sources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible [21].

Based on the mentioned researches, especially on the paper of [7-11], in this study a Distributed Knowledge-Based System for a grid-computing environment is proposed. The paper describes the synchronization concepts in ATFM, the main structure of the system, the relationships among the components of the model of

ATFM Grid Computing - ATFMGC, and the implementation of the prototype. At this moment, ATFMGC was implemented as a prototype in a network with three computers. As an example, a tactical planning case study related with some Brazilian airports is illustrated. As the preliminary study, the paper shows that the investigation of a Distributed Knowledge-Based System using grid-computing for real time traffic synchronization is not only a simple application, but it is an important study topic in proper AI. After deep research on this subject, the potential benefits might be significant for artificial intelligence, grid computing and air transportation.

The following sections are organized in five parts: soon after this introduction, section 2 presents the basic concept and characteristics about ATFM. In the third section, architecture and components of ATFMGC are described. Fourth section presents the implementation of ATFMGC, agents and inference processing. The case study is illustrated in fifth section. And finally, sixth section shows the conclusions.

2. ATFM Real - Time Air Traffic Synchronization Problem

ATFM is developed to ensure an optimum flow of air traffic to or through areas within which traffic demands exceeds the available capacity of the ATC system [8] at certain times. General scenery of ATFM is shown in figure 1. ATFM includes four main functions: Strategic Planning (involving long term: days to years), Pre-Tactical and Tactical Planning (involving middle term: from 1 hour to days), Short Term planning (involving short term: from 20 minutes to 1 hour), and Monitoring and Control (On-Line operation). As mentioned by [1], the ATFM system shows the following two special properties:

- Real-time Air Traffic Synchronization, an activity, which consists in implementing corrective actions on traffic applicable until the traffic is actually received by controllers to protect.
- Dynamic collaborative decision-making: aimed at achieving prompt dynamic “agreements” between Traffic Managers co-involved in the implementation of corrective actions on traffic transiting from one sector to the other.

In this study, the scope is limited to the ATFM tactical planning. The main functions are considered as following [7,9]:

- Creating a schedule for all departing and arriving flights to the airport, while maximizing the utilization of the runway and the terminal;

- Identifying congestion areas with regards of the air-space, aircraft and terminal constraints;
- Negotiating with other agents on the expected traffic flow to and from the airport;
- Re-schedule the flights according to the outcome negotiation; and
- Communicating the new schedule to other affected agents.

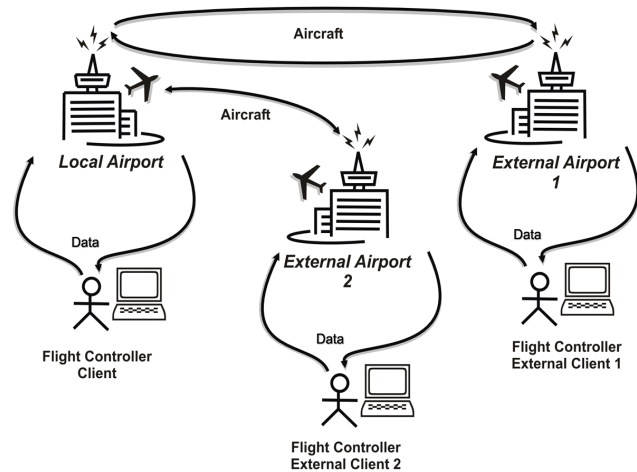


Figure 1. ATFM system

3. Architecture and Components of ATFMGC

Centralized Expert System for ATFM [7] has a simple structure and is easy to implement. This study uses the grid-computing platform to implement a distributed expert system on the grid computing.

3.1. Architecture

Grid computing distinguishes from conventional distributed computing by its focus on large-scale resource sharing, innovative applications and high-performance orientation. Grid technologies support flexible, safety, coordinated resource sharing among dynamic collections of individuals, institutions, and resources.

ATFMGC is proposed to support the sharing and coordinated using of diverse ATFM resources in dynamic and distributed Brazilian air traffic control system. As the airports are geographically distributed, components of ATFM operate by distinct controllers, some times with differing policies. ATFMGC in every airport is developed as a virtual computing system that is sufficiently integrated to deliver the desired quality of service over all ATFM.

Using web methods, the computational clients in ATFMGC can perform the computations on their own

airport data. The controller client does not connect directly to the “Final Service”, but connects to an Interface of web service. In this way, the program of controller clients is proposed for the complexity of parallel computation to resolve the ATC/ATM conflicts and insert the changes of flight schedule at each airport. As shown in figure 2, all the clients communicate the web methods and receive the results.

The proposed ATFMGC architecture consists of five components: Interface, Open Grid Services Architecture (OGSA), Web Services, Knowledge-Based System (KBS) and Database distributed in a set of airports. The basic components in the ATFMGC architecture and the relationship among the components are shown in Fig. 2.

3.2. Basic Components in ATFMGC

Interface. The function of operation interface is to help the flight controller to use ATFMGC through Web Services. Assuming that each airport has its own operation system, with this interface, the controller sends the requirements and receives the answers from related airports, through some specific notification services of grid computing.

OGSA Globus Toolkit. The Open Grid Services Architecture (OGSA) aims to define a new common and standard architecture for grid-based applications. Grid technologies, as Globus Toolkit, are evolving toward an Open Grid Services Architecture (OGSA) in which a Grid provides an extensible set of services that virtual organizations (in this research, ATFM system at each airport) can aggregate in various ways. Based on the concepts and technologies from both the Grid and Web services communities, OGSA defines a semantic uniform of service (the Grid service); defines standard mechanisms for creating, naming, and discovering transient Grid service instances; provides transparency of location and multiple protocol bindings for service instances; and supports integration with underlying native platform facilities. OGSA also defines, in terms of Web Services Description Language (WSDL) interfaces and associated conventions, mechanisms required for creating and composing sophisticated distributed systems, including lifetime management, change management, and notification. Service bindings can support reliable invocation, authentication, authorization, and delegation. In this work, the basic structure is to create the synchronism service among the involved airports. Java is defined by OGSA as a basic framework of Grid Service.

Web Services. This is an essential component for the implementation of ATFMGC. As expressed on [16,17], Web Services are the basis for Grid Services, which are

the cornerstones of the Globus Toolkit 3. Since they use standard XML languages, Web Services are both platform and language independent. Considering the fact that most Web Services use HTTP for transmitting messages (such as the service request and response), it represents an important advantage to build an Internet-scale ATFM application in the near future. Web services address heterogeneous distributed computing by defining techniques for describing software components, methods for accessing these components, and discovery methods that enable the identification of relevant service providers. A key advantage of Web services is their programming language, model, network, and system software neutrality.

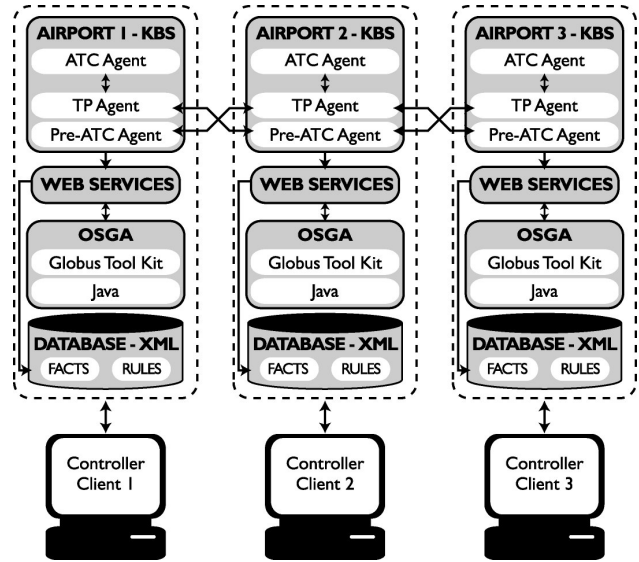


Figure 2. ATFMGC architecture

Agents and Knowledge Based System (KBS). KBS contains both domain knowledge (facts and rules) and the processes of structuring knowledge (knowledge representation) and is distributed at each airport in ATFMGC. The facts represent the local and real time updated information. Rules and the processes of structuring knowledge are almost the same at each airport. Each referring airport possesses its own Knowledge Base constituted of a variety of functional agents such as Pre-ATC agent and Tactical Planning (TP) agent etc.

Databases at each airport consist of three parts: flight timetable, on-line traffic information and database administration. Due to the real time traffic synchronization property, the database systems should be easy and fast accessed. All the data such as aircraft and runway condition are stored in the database with XML form. The database is constructed through JDOM parser, an open source, tree-based, pure Java API.

Parameters need to be considered in using a Grid as following [21]. Some of them are still in development in this research.

4. Inference Process and Implementation

In this study, Pre-ATC agent and Tactical Planning (TP) agent are developed in ATFMGC. The former article is an expert system for local air traffic control, which is designed just to simulate the situation for ATFM purpose. The later is a distributed KBS for Tactical Planning of ATFM. ATC agent is also mentioned in this research but is not described in detail. Figure 3 shows the manner of communication and negotiation among agents in ATFMGC system.

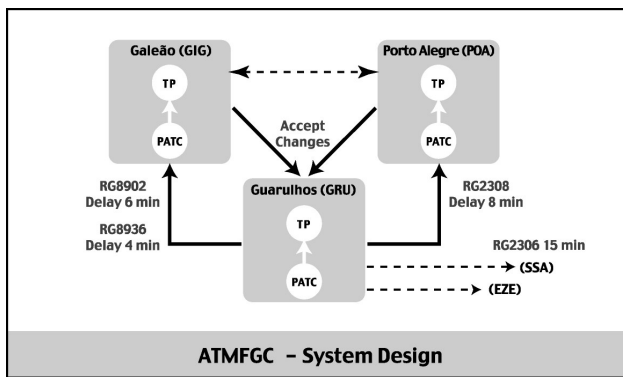


Figure 3. Communication and Negotiation among Agents

4.1. Pre-ATC Agent

Pre-ATC agent of ATFMGC is represented according to ATC/ATM rules, which have been defined by the Brazilian Department of Civil Aviation - DAC [7, 20]. Six processes have been developed for the knowledge representation of Pre-ATC agent: flight data, scheduling and control condition, separation standards, holding assignment, departure and arrival conflict prevention and take-off delay assignment [7,8]. There are three types of ATC rules: for landing aircraft, both for landing and take-off aircraft and for take-off aircraft.

4.2. Tactical Planning Agent

To measure the traffic congestion, Weighted Combined Total Delay (WCTD) is defined as basic index [9]. It uses a different delay cost function for the landing and taking off of flights. Considering that the objective of ATFM is to minimize the aircraft air holding, in this research the delay cost function is simplified to multiply every minute of delay by a weight. For landing flight, the weight is chosen as 5, and take-off the weight is as 1.

4.3 Inference Process

The main inference process Pre-ATC agent is the same as [7,8]. The following steps are inference processes of TP agent:

1. Web Services component of TP agent at airport A is used to communicate with Pre-ATC agent at airport B (and others) to get the departure flight delay requirements at airport A.
2. Schedule process of TP agent at airport A reschedules the departure flights according to the accepted WCTD value at A.
3. Diagnosis process checks the new schedule and stores it in the database at airport A if there is no conflict. When any conflict takes place, the Schedule process repeats the work again.
4. Broadcasts and Negotiation generates a message to Web Services to send to the TP agents at related airports.
5. At the same time, ATFMGC at related airport is also working with the changed schedule.
6. If the system detects the conflicts at a related airport, a new schedule is generated at other airports and the information is sent back to airport A. Broadcasts and Negotiation process negotiates with the related airport.
7. Evaluation and Validation processes verify the actual schedule within a certain time period.
8. The PT agent at airport A is invoked recursively until there are no more conflicts.

4.3. Implementation

At this moment, a prototype of ATFMGC has been developed using Globus ToolKit (version Alpha 3) on a grid computing. The prototype consists of three personal computers (Pentium 4). Each computer with ATFMGC interface represents the ATFM operation system of an airport. The whole system was codified with JAVA language.

The KBS, which consists of the rules and facts was developed using XML language. XML documents use parser JDOM library to for the interpretation of the stored data. Web Services and client interface also have been integrated with ATFMGC in JAVA. Apache TomCat 4.0 is used as Servlet Container. For the installation and manipulation of the Globus ToolKit, Cygwin is chosen as emulator.

5. An Illustrative Example

When executing an ATFMGC simulation, flight information was collected from 9:00 to 10:00 am at February 16th, 2004 [19]. The simulation started at 7:00

am (Current Time). The Pre-ATC agent (PATC) in Guarulhos International Airport (GRU) began to estimate the air traffic congestion in a period of time Δt_2 , from t_1 to t_2 , where $t_1 = \text{Current time} + \Delta t_1$, $t_2 = t_1 + \Delta t_2$. Defining $\Delta t_1 = 2$ hours and $\Delta t_2 = 1$ hour, the Pre-ATC agent will consider the period $t_1 = 9:00$ and $t_2 = 10:00$ hours. Some arrival and departure flights during this period are shown in Tables 1 and 2.

Table 1. Arrival Flights to Guarulhos Airport

Flight	Type	Departure Airport	Planned Departure Time	Planned Arrival Time	Confirmed Arrival Time	Delay σ_1	Planned Departure Change	Time Slice
RG8631	738	EZE	06:30	09:05	09:05			TS2
UA861	763	IAD	21:30	09:10	09:10			TS3
RG2209	733	BSB	07:35	09:15	09:17	00:02		TS4
RG8920	M11	GIG	08:15	09:15	09:15			TS4
PU222	73S	MVD	07:00	09:20	09:22	00:02		TS5
RG2308	735	POA	07:50	09:20	09:28	00:08	07:58	TS5
RG8902	733	GIG	08:20	09:20	09:26	00:06	08:26	TS5
RG8936	738	GIG	08:20	09:20	09:24	00:04	08:24	TS5
AA995	777	MIA	23:20	09:20	09:20			TS5
AR1240	73S	EZE	06:50	09:37	09:37			TS8
RG2331	735	SSA	07:15	09:40	09:40			TS9

To detect the aircrafts traffic congestion, time period Δt_2 is divided in Time Slices (TS) and the traffic congestion is calculated in each slice. Each TS is selected with duration of 5 minutes. The Pre-ATC is used to calculate the WCTD for the flights in tables 1 and 2 from 9:00 to 10:00. First, a WCTD value of 50 is an accepted holding delay. Figure 4 illustrates the WCTD values at GRU.

Only at TS5 from 9:20 to 9:25, the WCTD was greater than the acceptable WCTD value, as shown above. At other TSs, including the TS4 and TS7 whose numbers of flights exceeds the airport capacity, the WCTDs values were still acceptable. Local ATC agent solved the conflicts in all TSs, except TS5, according to ATC/ATM rules. Some flights were delayed en-route and others on ground. New schedule is presented in columns 6 and 7 of tables 1 and 2. The flights in TS5 were re-scheduled by the negotiation with other airports.

There are 5 arrivals at 9:20 am of TS5: PU222 is from Montevideo (MVD), RG2308 is from Porto Alegre (POA), RG8902 and RG8936 are from Galeão (GIG) and AA995 is from Miami (MIA). In order to reduce the time the aircraft will be holding in the air, some of them had been kept on ground at their original airports. An aircraft can be delayed on ground in the original airport, whenever there is a request from any other airport in a given time before the planned schedule time for take-off. This time was called MINTIME [9] and was defined, in this research, as 40 minutes.

Table 2. Departure Flights from Guarulhos Airport

Flight	Type of Aircraft	Arrival Airport	Planned Departure Time	Planned Arrival Time	Confirmed Departure Time	Delay σ_2	Time Slice
JJ3315	100	CPQ	09:15	09:45	09:18	00:03	TS4
JJ3811	100	CWB	09:15	10:10	09:19	00:04	TS4
JJ8001	320	EZE	09:15	12:15	09:29	00:14	TS4
RG2306	733	SSA	09:15	11:35	09:30	00:15	TS4
JJ3342	100	CNF	09:25	10:30	09:31	00:06	TS6
JJ3490	320	NAT	09:30	13:00	09:32	00:02	TS7
JJ3506	320	REC	09:30	12:40	09:33	00:03	TS7
JJ3804	320	CGR	09:30	10:10	09:34	00:04	TS7
RG3490	320	NAT	09:30	13:00	09:35	00:05	TS7
RG8880	733	VVI	09:35	11:45	09:36	00:01	TS8
PZ0707	100	ASU	09:40	11:30	09:43	00:03	TS9

Because its condition, the international flight AA995 from Miami was arranged to land at 9:20 am and due to its priority, it was impossible to change the situation. The flight PU222 from Montevideo, was kept in the air for 2 minutes and the land schedule at 9:22 am. Concerning the others flights, the Pre-ATC negotiated with other Tactical Planning agents at original airports. As a result of the negotiation, it was considered to delay on ground the flight RG8902 in 6 minutes and RG8936 in 4 minutes, both at Galeao International Airport (GIG). The flight RG2308 was proposed to delay on ground 8 minutes at Porto Alegre International Airport (POA).

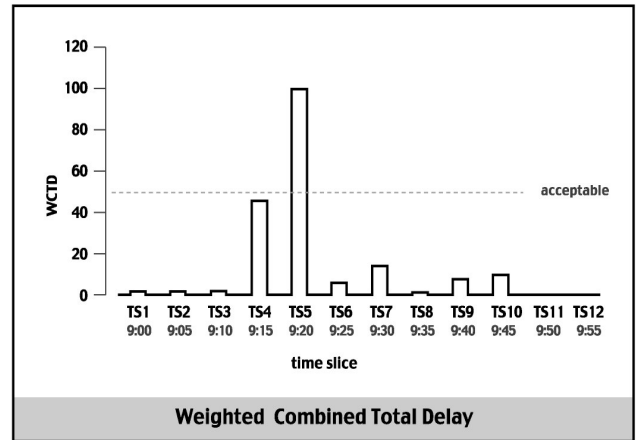


Figure 4. WCTD values at GRU from 9:00 am to 10:00 am

At the same time, the Pre-ATC agents at GIG and POA also verified occasional conflicts and may ask for flight delays at other airports. When the TP agent at GIG received the request for the delay of flights RG8902 and RG8936 from Pre-ATC agent at GRU, the conflict analysis process was repeated again and again. In spite of enhances the WCTD, this change in the flight schedule is

still possible because the value remains acceptable due to the flights RG8936 and RG8902.

The TP agent at GRU might also communicate to the agents in the destination airports in the case there are significant delays of flight departures, which is defined as σ_2 . As shown in table 2, any flight whose σ_2 is greater than 10 minutes, this delay is informed to the destination airport. In this case, the flight delay of RG2306 was informed to TP agent at Salvador airport (SSA).

In this way, the TP agents of different airports involved in this study coordinate their processes until the whole ATFMGC net achieved synchronism. In the case it does not happen, a maximum time ρ shall not be exceeded. The value of ρ means the maximum time that the Pre-ATC can wait to receive the request confirmation.

6. Conclusions

Using grid computing to the Real-Time Traffic Synchronization problem, especially ATFM, was proposed in this research. ATFM is an interesting domain for the application of distributed Knowledge based system. Any study about distributed Knowledge based system in real time air traffic synchronization problem using grid computing hasn't ever been reported until now. Bearing in mind the advantages of grid computing, this proposal also presents a solution for air transportation. The result obtained in this work may represent a significant contribution to the research of artificial intelligence, grid computing and air traffic transportation [12,18]. For further study, the following aspects may be taken into deep consideration:

- Constructing an special grid computing for ATFM, to study the main components (both hardware and software), and relationship among computing and air traffic control;
- Parameters to organize the rescheduling among the grid nodes (airports) are still in the development. They are still important for the further evaluation of the system;
- Implementing the ATFMGC to the whole Brazilian ATC/ATM system in order to make ATFM tactical planning and real time control.

References

- [1] S. Stoltz, R. Guerreau - EUROCONTROL, "Future ATFM Measures (FAM) operational Concept", EEC Note No. 13/02, 2002.
- [2] S. Stoltz and P. Ky, "Reducing Traffic Bunching More Flexible Air Traffic Flow Management", 4th USA/Europe ATM R&D Seminar, New_Mexico, 2001.
- [3] D. Dippe, "4D-Planner – A Ground Based Planning System for Time Accurate Approach Guidance, DLR- Mitt, 1989, 89-23.
- [4] U. Völckers, "Approach Towards a Future Integrated Airport Surface Traffic Management", DLR- Mitt, 1989, 89-23.
- [5] G. D. Gosling, "Application of Artificial Intelligence Application in Air Traffic Control", Trans. Res., 21A(1), 1987.
- [6] U. R. Schlatter, "Real Time Knowledge Based Support for Air Traffic Flow Management", IEEE Expert, 1994, pp. 21-24.
- [7] L. Weigang, C. J. P. Alves and N. Omar, "An expert system for Air Traffic Flow Management". Journal of Advanced Transportation", Vol. 31, No. 3, 1997, pp. 343-361.
- [8] L. Weigang, C. J. P. Alves and N. Omar: Knowledge - Based System for Air Traffic Flow Management: Timetable Rescheduling and Centralized Flow control, Applications of AI in Engineering VIII, Vol. 2, G. Rzevski, J. Pastor, and R. A. Adey (Eds.), Elsevier, (1993), 655-670.
- [9] G. Tidhar, A. Rao, M. Ljunberg, "Distributed Air Traffic Management System", Technical note, No. 2, 1992.
- [10] T. Prevôt, "Exploring the Many Perspectives of Distributed Air Traffic Management: The Multi Aircraft Control System MACS", in S. Chatty, J. Hansman, and G. Boy (Eds.), Proceedings of the International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero 2002), AAAI Press, Menlo Park, CA, pp. 149-154.
- [11] M. Nguyen-Duc, J.-P. Briot, Alexis Drogoul, V. Duong, "An application of Multi-Agent Coordination Techniques in Air Traffic Management", in the Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology (IAT'03), 2003.
- [12] J. Cao, S. A. Jarvis, S. Saini, D. J. Kerbyson and G. R. Nudd: ARMS: an Agent-based Resource Management System for Grid Computing, Scientific Programming, vol. 10, (2002), 135-148.
- [13] F. Berman, G. Gox, T. Hey: Grid Computing: Making The Global Infrastructure a Reality, John Wiley & Sons, (2003).
- [14] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, N. Bieberstein: Introduction to Grid Computing with Globus, IBM, (2003).
- [15] G. J. Portella, A. C. M. A. Melo: A Load Balancing Strategy to Schedule Independent Tasks in a Grid Environment, to appear at Europar04, (2004).
- [16] B. Sotomayor: The Globus Toolkit 3 Programmer's Tutorial, (2003).
- [17] I. Foster and C. Kesselman: The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publisher (1999).
- [18] G. Weiss (Ed): Multiagent systems, The MIT Press, (2000).
- [19] Panrotas Editora Ltda, "Guia de Horário de Nacionais e Internacionais", No. 370, 2004.
- [20] DEPV, IMA 100-12, portaria DEPV No. 46 de 30/06/99, Departamento de Previsão de Voo – DEPV, Ministério de Aeronáutica, 1999.
- [21] Nabrzyski, J. M. Schopf, J. Weglarz (Eds.): Grid resource management - State of the Art and Future Trends, Kluwer Academic Publishers, (2003).

Extracting Minimal Non-Redundant Implication Rules by Using Quantized Closed Itemset Lattice

LI Yun^{1,2} LIU Zongtian¹ CHENG Wei² WU Qiang¹ LIU Wei¹

¹ School of Computer Science, Shanghai University, Shanghai, 200072, China

² School of Information Engineering, Yangzhou University, Jiangsu, 225009, China

E-mail: yzliyun@163.com

Abstract: The association rules are usually extracted from frequent itemsets(FIs), but the number of FIs is enormous, and many redundant rules exist in the mined rules. The frequent closed itemsets are adopted in order to reduce the number of FIs. With the inherent closure properties in objects and attributes, concept lattice is very suitable for representing the relation between closed itemsets. In this paper, the quantized closed itemset lattice is formed with the modified node structure. As well, a scheme for quantized closed itemset lattice built incrementally from database is adopted. Extracting minimal non-redundant rules decreases the number of rules without reducing any useful information. The implication rule is exact, and an innovative algorithm for extracting such rule is presented, which can directly extract minimal non-redundant implication rules from the quantized closed itemset lattice.

Keywords: Frequent itemset, Frequent closed itemset, Quantized closed itemset lattice, Implication rule.

1. Introduction

The association rule[1] is the main form of knowledge extracted from database, and is usually extracted from frequent itemsets(FIs), but their number is enormous. In order to reduce the number of FIs without losing any useful information, frequent closed itemsets(FCIs) [2,3,4] are adopted to extract association rules.

A rule is described as the relation between intension (attribute) sets, and reveals the inclusion relation between extension (object) sets. In a concept lattice[5], a node consists of intension and extension, and the relation between nodes reflects the generalization and specialization relationship between concepts. With the inherent closure properties in objects and attributes, concept lattice is very suitable for representing the relation between closed itemsets.

Some algorithms such as CLOSE[3], CHARM[6], CLOSET[7] are used for mining FCIs. Both CLOSE and

CHARM need to scan database more than once and generate candidate itemsets, and must regenerate the whole FCIs when new transactions are inserted into database. Although CLOSET algorithm needs to scan database only once and doesn't need to generate candidate itemsets, it also doesn't fit for the dynamic databases. An approach for generating the FIs incrementally has been presented in the literature [2]. If the database is updated, only parts of the FIs need to be updated. This approach utilizes the method of concept lattice for incremental updating.

The closed itemset lattice can be formed by closed itemsets using the Galois connection. In this paper, we adopt the incremental approach to build the lattice, and modify the lattice node. This modified lattice is called quantized closed itemset lattice.

According to the confidence, association rules can be classified into exact and approximate. Exact association rules can also be named as implication rules, which have 100% confidence. Many redundant rules are obtained when rules are extracted from context. A minimal non-redundant rule is one with minimal antecedent and maximal consequent, and the other rules can be deduced by a set of these rules. An algorithm for extracting such rules was presented by Y. Bastide et al [8], but it generates the FCIs by using CLOSE algorithm.

In this paper, we adopt a scheme for building a quantized closed itemset lattice incrementally from a database, and an innovative approach for extracting implication rules is presented, which can directly extract minimal non-redundant implication rules from the quantized closed itemset lattice.

2. Basic Notion of Association rule

Let $I=\{i_1, i_2, \dots, i_m\}$ be a set of items, D be a transaction database. A transaction T is a set of items in I , and has a unique identifier called TID or tid. A sample of transaction database is shown in Table 1. The proportion of transactions in D that contain an itemset X is called the support of X and is denoted $\text{sup}(X)$. An itemset X is

frequent when $\text{sup}(X)$ reaches at least a user-specified minimum threshold called *minsup*, i.e. $\text{sup}(X) \geq \text{minsup}$.

An association rule R is an expression $X \Rightarrow Y$, where $X, Y \in I$ and $X \cap Y = \Phi$. The support of rule R is defined as $\text{sup}(X \cup Y)$ while its confidence $\text{conf}(R)$ is computed as the ratio $\text{sup}(X \cup Y) / \text{sup}(X)$. A rule R is confident when $\text{conf}(R)$ reaches at least a user-specified minimum threshold called *minconf*, i.e. $\text{conf}(R) \geq \text{minconf}$.

Table 1 A sample of transaction database

TID	items
1	A, C, D
2	B, C, E
3	A, B, C, E
4	B, E
5	A, B, C, E
6	B, C, E

The association rule extracted from D is a valid rule if its support and confidence are more than or equal to a user-specified *minsup* and *minconf*.

A rule $X \Rightarrow Y$ is called exact association rule if its confidence reaches 100%, also referred to as an implication rule in this paper.

Definition 1 Let X, Y be two frequent itemsets, $X, Y \subseteq I$, and their supports be $\text{sup}(X)$ and $\text{sup}(Y)$ respectively. If $Y \subset X$, $Y \neq \Phi$ and $\text{sup}(X) = \text{sup}(Y)$, then $Y \Rightarrow X - Y$ is an implication rule.

As for the transaction database shown in Table 1, if *minsup* = 2/6, all frequent itemsets form a lattice shown on the left of Fig.1 or Fig. 1(left). Each node in the lattice is a pair of a frequent itemset and its corresponding tidset, i.e. (tidset, itemset).

3. Frequent Closed Itemset and Quantized Closed Itemset Lattice

Fig. 1(left) shows that there are some different FIs with the same tidset in the set of FIs.

Definition 2 Let (X, Y) be a pair of the frequent itemset Y and corresponding tidset X . The frequent itemsets with the same tidset X is referred to as the same tidset's

frequent itemsets, these itemsets and tidset X form the Set of Pairs of the Same Tidset's Frequent Itemsets, denoted SPSTFI.

Some SPSTFIs in Fig. 1(left) are shown in different fonts. For example, there is a SPSTFI with the tidset 35 (it actually stands for {3,5}), and similarly, ABC for {A,B,C}), this SPSTFI with the tidset 35 is denoted as SPSTFI_{35} , and $\text{SPSTFI}_{35} = \{(35, ABCE), (35, ABC), (35, ABE), (35, ACE), (35, AB), (35, AE)\}$, shown with boldface type.

Definition 3 Let (X, Y) be an element of a SPSTFI. If there is not another element (X, Y') such that $Y' \supset Y$, then (X, Y) is defined as the largest pair of the tidset and itemset, and Y is a largest frequent itemset; Similarly, if there is not another element (X, Y') such that $Y' \subset Y$, then (X, Y) is defined as the least pair of the tidset and itemset, and Y is a least frequent itemset.

For instance, in above SPSTFI_{35} , $(35, ABCE)$ is the largest pair, and $ABCE$ is the largest frequent itemset; $(35, AB)$ and $(35, AE)$ are the two least pairs, AB and AE are the two least frequent itemsets

Theorem 1 If (X, Y) is the largest pair of the tidset and itemset in a SPSTFI, then it is unique.

Proof: Suppose there is another largest pair (X, Y') in the SPSTFI besides (X, Y) , and let $(X, Y) = (X, Y_1 Y_2 \dots Y_k Y_{k+1} \dots Y_m)$ and $(X, Y') = (X, Y_1 Y_2 \dots Y_k Y'_{k+1} \dots Y'_n)$. According to the definitions of the pair of the tidset and itemset, the tidset X contains the itemset $Y_1 Y_2 \dots Y_k Y_{k+1} \dots Y_m$ and $Y_1 Y_2 \dots Y_k Y'_{k+1} \dots Y'_n$, therefore, the tidset X certainly contains the itemset $Y_1 Y_2 \dots Y_k Y_{k+1} \dots Y_m Y'_{k+1} \dots Y'_n \supset Y_1 Y_2 \dots Y_k Y_{k+1} \dots Y_m$. It conflicts with that (X, Y) is a largest pair.

For the space limitation, the proof of other theorems is omitted.

Not only are these frequent itemsets with same tidset unable to convey new useful information but they also increase the complexity of lattice. Therefore, the frequent closed itemset (FCI) is adopted.

A transaction database can be easily denoted as a formal context which is a triple $D = (O, A, R)$, where O is the set of objects, corresponding to the tids in the transaction database, and A is the set of attributes,

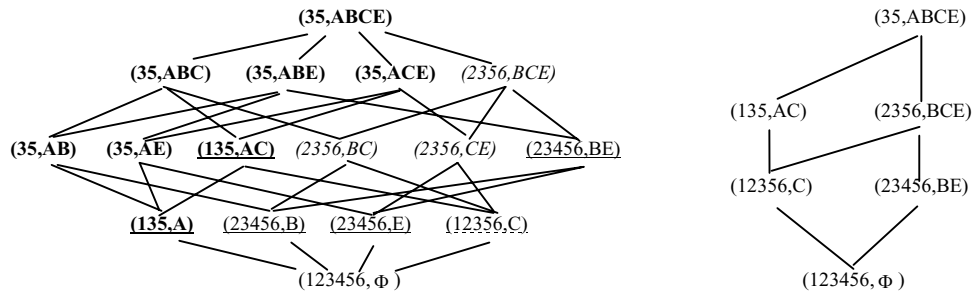


Fig.1 The lattice of all frequent itemsets with support $\geq 2/6$ and its corresponding frequent closed lattice of the sample database

corresponding to the items. $R \subseteq O \times A$ is a binary relation, For an object $o \in O$, an attribute $a \in A$, then oRa means that transactions o has item a .

The object set $X \in P(O)$ and the attribute set $Y \in P(A)$ are connected with the relation as follows:

$f(X) = \{y \in A \mid \forall x \in X, xRy\}$, $g(Y) = \{x \in O \mid \forall y \in Y, xRy\}$. This connection is Galois connection. As for a binary tuple (X, Y) , $X = g(Y)$ is the tidset which contains the itemset Y and $Y = f(X)$ is the itemset that is contained in all the transactions in X . A pair (X, Y) of the tidset and itemset is known as an itemset concept.

If itemset concepts $C_1 = (X_1, Y_1)$, $C_2 = (X_2, Y_2)$ satisfy $Y_1 \subseteq Y_2$, then (X_1, Y_1) is called child concept and (X_2, Y_2) is called parent concept, and the sub-super relation can be denoted by $(X_1, Y_1) \leq (X_2, Y_2)$. If there is not $C_3 = (X_3, Y_3)$ such that $(X_1, Y_1) < (X_3, Y_3) < (X_2, Y_2)$, then (X_1, Y_1) is direct child concept and (X_2, Y_2) is direct parent concept.

Definition 4 Let $D = (O, A, R)$ be a transaction database or a formal context, Y is an itemset and $Y \in A$, then Y is referred to as Closed Itemset denoted CI if $Y = f(g(Y))$, and its support $\text{sup}(Y) = |g(Y)| / |O|$.

Definition 5 If Y is a closed itemset, and its support is not less than minsup , then Y is referred to as Frequent Closed Itemset denoted as FCI.

A pair $(g(Y), Y)$ is an itemset concept composed of a frequent closed itemset Y and its corresponding tidset $g(Y)$. These Pairs of FCI and its corresponding tidset can consist of the frequent closed itemset lattice. For example, such lattice of the sample database is shown on the right of Fig.1 where $\text{minsup} = 2/6$. It can be found out that the number of FCIs is much smaller than that of FIs. Actually, a SPSTFI is associated with one FCI, and that, the pair of this FCI and its corresponding tidset is the largest pair of the tidset and itemset in the SPSTFI.

Theorem 2 Let Y be a largest frequent itemset in a SPSTFI, then it is certainly a frequent closed itemset, and vice versa.

Since the transaction database is frequently updated, there is a need to keep the whole set of CIs including those which are not frequent. The closed itemset lattice can be built incrementally. While database is updated, it is only to make the parts of nodes in the lattice update. For extracting rules conveniently, the node structure in the lattice is modified and it includes some components as follows: $(\text{Itemset}, \text{Tid_count}, \text{Dirt_parents}, \text{Dp_count}, \text{Dirt_childs}, \text{Dc_count})$ where Itemset is the itemset contained in the node, Tid_count is the cardinality of tidset, Dirt_parents and Dp_count are the node's direct parent concepts and their number, Dirt_childs and Dc_count are the direct child concepts and their number. The closed itemset lattice with these node structure is known as the Quantized Closed Itemset Lattice denoted QCIL, and its building algorithm is similar to that in our previous work reported in [9]. Because of the brief length of this paper, the algorithm isn't presented here.

4. Extraction of Minimal Non-redundant Implication Rules

Among the rules extracted from FIs with the same support and confidence, some can't give any additional information, that is to say, they are redundant rules. For example, $ab \Rightarrow cde$, $ab \Rightarrow c$, $ab \Rightarrow d$, $abc \Rightarrow d$, $abcd \Rightarrow e$, $abde \Rightarrow c$ are six valid rules with the same support and confidence. But compared with the first rule, the other five rules don't give additional information. They are redundant to the first rule $ab \Rightarrow cde$, and the rule $ab \Rightarrow cde$ is that with minimal antecedent and maximal consequent.

It can be found out that the implication rule $Y \Rightarrow X-Y$ certainly exists where X, Y are itemsets in a SPSTFI and $Y \subset X$, $Y \neq \Phi$. The rules extracted from a SPSTFI have the same support and confidence. Therefore, with a SPSTFI, it is enough to extract the implication rule with minimal antecedent and maximal consequent.

Definition 6 An implication rule $X \Rightarrow Y$ is a minimal non-redundant implication rule iff there doesn't exist another rule $X' \Rightarrow Y'$ with $\text{sup}(X) = \text{sup}(X')$, $X' \subseteq X$ and $Y \subseteq Y'$.

Evidently, if Y is a least frequent itemset and X is the largest frequent itemset in a SPSTFI, then $Y \Rightarrow X-Y$ induced by itemsets X and Y is certainly a minimal non-redundant implication rule. For example, $ABCE$ is the largest FI and AB, AE are two least FIs in SPSTFI_{35} , so there exist two minimal non-redundant implication rules $AB \Rightarrow CE$, $AE \Rightarrow BC$.

The QCIL can be directly built in an incremental manner from the context $D = (O, A, R)$. According to Theorem 2, the node C in QCIL is certainly the largest pair in its corresponding SPSTFI denoted as SPSTFI_C if $\text{sup}(C) = \text{Tid_count}(C) / |O| \geq \text{minsup}$ and $|\text{Itemset}(C)| \geq 2$, and if only the C' , the least pair of tidset and itemset in SPSTFI_C is found, the minimal non-redundant implication rule can be obtained. The condition $|\text{Itemset}(C)| \geq 2$ is guaranteed to extract the rule with non-null consequent.

Theorem 3 Let node C in QCIL be frequent, then the other pair C_1 in the SPSTFI_C satisfies $\text{Itemset}(C_1) \subset \text{Itemset}(C)$ and $\text{Itemset}(C_1) \not\subseteq \text{Itemset}(\text{Dirt_childs}(C))$.

According to the theorem 3, the method for searching the least itemsets among the corresponding SPSTFI_C of node C in QCIL is to examine whether the elements (itemsets) of the powerset of $\text{Itemset}(C)$ are contained in the itemsets of $\text{Dirt_childs}(C)$ in the ascending order of the element cardinality, and then get the least itemsets which are the least elements not in the itemsets of $\text{Dirt_childs}(C)$. Furthermore, the cardinality of the least itemset is at most the number of $\text{Dirt_childs}(C)$, which can be used as the terminal condition of search procedure, and will greatly reduce the time for searching the set of the least pairs of the tidset and itemset.

Using above method, the algorithm for extracting such implication rules is described as follows:

Algorithm 1:extracting the minimal non-redundant implication rules

INPUT:the quantized closed itemset lattice L, minimal support *minsup*

OUTPUT: the set of minimal non-redundant implication rules MIR

MIR= Φ

FOR each node C in L in ascending $|Itemset(C)|$ **DO**

IF $|Itemset(C)| \geq 2$ and $Tid_count(C)/|O| \geq minsup$ **THEN**

MCS= Φ ; it is the set of least pairs in SPSTFI_C

FOR each non-null Ck $\in P(Itemset(C))$

in ascending $|Ck|$ **DO**

IF $|Ck| > Dc_count(C)$ **THEN** exit **FOR** **ENDIF**

FOR each Cp $\in Dirt_childs(C)$ **DO**

flg=true

IF Ck $\subset Itemset(Cp)$ **THEN** flg=false;

exit **FOR** **ENDIF**

ENDFOR

IF flg and no such Ck' itemset in MCS
that Ck' $\subset Ck$ **THEN**

MCS= MCS $\cup (Tid_count(C), Ck)$

ENDIF

ENDFOR

FOR each C' \in MCS **DO**

MIR=MIR $\cup \{Itemset(C') \Rightarrow Itemset(C) \setminus Itemset(C'),$
sup= $Tid_count(C)/|O|\}$

ENDFOR

ENDIF

ENDFOR

If *minsup*=2/6, the minimal non-redundant implication rules is shown in Table 2 by applying Algorithm 1 to the QCIL built from the sample database shown in Table 1. The result is identical with that obtained by using CLOSE algorithm in the literature [8].

Table 2 Minimal non-redundant implication rules extracted from the sample database

MIR	support
A \Rightarrow C	3/6
B \Rightarrow E	5/6
E \Rightarrow B	5/6
AB \Rightarrow CE	2/6
AE \Rightarrow BC	2/6
BC \Rightarrow E	4/6
CE \Rightarrow B	4/6

5. Conclusion

The CI is the subset of the itemset, and the FCI is the subset of CI. The number of FCIs is much less than that of FIs and without reducing any useful information. Extracting the rule from the FCIs will decrease the search space and benefit to reduce the complexity of the rule

extraction. The QCIL is the closed itemset lattice with modified node structure. The scheme of building the QCIL incrementally from database will take only one pass over the database and only parts of the nodes in this lattice need updated when new transactions are inserted into the database, and it is specially suitable for managing the dynamic database, but other algorithms such as CLOSE, CHARM, CLOSET don't fit for the dynamic databases. The innovative algorithm can be applied to directly extract minimal non-redundant implication rules with minimal antecedent and maximal consequent from the QCIL.

Acknowledgment

This work is supported by National Natural Science Foundation of China, No. 60275022

References

- [1] Han J., Kamber M, "Data Mining: Concepts and Techniques", Morgan Kaufmann Publisher, 2000
- [2] Petko Valtchev, Rokia Missaoui, Robert Godin, Mohamed Meridji, "Generating frequent itemsets incrementally: two novel approaches based on Galois lattice theory", Journal of Experimental and Theoretical Artificial Intelligence 14(2-3), 2002, pp.115-142.
- [3] N. Pasquier, Y. Bastide et al, "Efficient Mining of Association Rules Using Closed Itemset Lattices", Information Systems, 24(1), 1999, pp.25-46.
- [4] M.J. ZaKi, "generating Non-Redundant Association Rules", In proceedings of the 6th international Conference on Knowledge Discovery and Data Mining, 2000, pp. 24-43
- [5] B. Ganter, R. Wille, "Formal Concept Analysis: Mathematical Foundations", Springer-Verlag, 1999.
- [6] M.J. ZaKi, C.J. Hsiao, "CHARM: An Efficient algorithm for Closed Association Rule Mining", Technical report 99-10, Rensselaer Polytechnic, 1999.
- [7] J. Pei, J. Ha and R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", In proceedings of the ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp.21-30
- [8] Y. Bastide, N. Pasquier et al, "Mining Minimal Non-Redundant Association Rules using Frequent Closed Itemsets", Lecture Notes in Computer Science, vol 1861, 2000, pp.972-986.
- [9] Xie Zhi-peng, Liu Zong-tian, "Concept Lattice and Association Rule Discovery", Journal of Computer Research and Development, Vol 37, No.12, 2000, pp.1415-1421 (in chinese)

Formal Description Techniques for CSPs and TCSPs

Malek Mouhoub, Samira Sadaoui and Amrudee Sukpan
Department of Computer Science, University of Regina
3737 Waskana Parkway, Regina SK, Canada, S4S 0A2
{mouhoubm,sadaouis,supkan1a}@cs.uregina.ca

Abstract

LOTOS is a formal specification technique for describing and verifying complex systems. In this paper, we investigate the applicability of LOTOS to specify and solve Constraint Satisfaction Problems (CSPs) as well as Temporal Constraint Satisfaction Problems (TCSPs). A CSP is a general framework used to represent and solve a large variety of combinatorial problems including frequency assignment, configuration and conceptual design, network management and transportation. A TCSP is one particular case of CSPs, where constraints are temporal relations between temporal variables defined over a set of time intervals. TCSPs are used to handle problems involving temporal constraints such as scheduling, planning and computational linguistics. Through simulation and model-checking verification, we show, in this paper, how to solve CSPs and TCSPs using LOTOS specifications.

1. Introduction

LOTOS [BB89] is a formal specification technique for describing and verifying complex systems. In this paper, we investigate the applicability of LOTOS to specify and solve Constraint Satisfaction Problems (CSPs) [Mac77, HE80, Kum92] as well as Temporal Constraint Satisfaction Problems (TCSPs) [All83, Mei96, DMP91, vB92, MCH98]. A CSP is a general framework used to represent and solve a large variety of combinatorial problems including frequency assignment, configuration and conceptual design, scheduling and planning. A CSP involves a list of variables defined on finite domains of values and a list of relations restricting the values that the variables can simultaneously take. If the relations are binary we talk about binary CSPs. Solving a CSP consists of finding an assignment of values to each variable such that all relations (or constraints) are satisfied. A CSP is known to be an NP-Hard problem. Indeed, looking for a possible solution to a CSP requires a backtrack search algorithm of exponen-

tial complexity in time ¹. The backtrack search algorithm is a depth first search technique that incrementally attempts to extend a partial solution toward a complete one by repeatedly choosing a value for another variable. If a partial solution violates any of the constraints, backtracking is performed to the most recently instantiated variable that still has alternatives available.

In order to deal with problems involving numeric and symbolic temporal information, we have developed the model TemPro [MCH98], extending the interval algebra defined by Allen [All83] to handle numeric constraints. TemPro transforms any problem under qualitative and quantitative constraints into a binary CSP where constraints are disjunctions of Allen primitives [All83] and variables, representing temporal events, are defined on domains of time intervals. We call this later a Temporal Constraint Satisfaction Problem (TCSP) ².

LOTOS (Language of Temporal Ordering Specification) [ISO87, BB89] is the ISO standardized formal specification technique to describe and verify concurrent and open distributed systems. LOTOS has also been widely applied to other applications, such as bus architecture, conformance testing, computer integrated manufacturing, and distributed transaction processing. LOTOS combines a process calculus with a data type language. A data type identifies a set of values or domains, a set of associated operations, and a set of equations. Equations are equalities between terms. With the data part we can specify the different constraints of a CSP, their corresponding variables and domains, as well as the temporal constraints of a TCSP. In the other hand, the process part, describing processes or behavior expressions, defines the external behavior of a system. In CSPs and TCSPs, it corresponds to the description of the resolu-

¹Note that some CSPs can be solved in polynomial time. For example, if the constraint graph corresponding to the CSP has no loops, then the CSP can be solved in $O(nd^2)$ where n is the number of variables of the problem and d is the domain size of the different variables.

²Note that this name and the corresponding acronym was used in [DMP91]. A comparison of the approach proposed in this later paper and our model TemPro is described in [MCH98].

tion process, such as the constraint propagation and backtracking algorithms. Behavior expressions are built using LOTOS operators such as the action prefix $;$ which denotes a sequence of actions, and **exit** the successful termination of a specification. An action can be followed by the construct $?$ in order to input values from the environment, e.g. *Input?X: color* expresses that the user can enter a value of type color. In LOTOS, the unary constraints are defined using the selection predicates $[]$ in order to restrict the values offered within an action, e.g. *Input?X:natural [X<4]* means that the domain of the variable X is restricted to the set $\{1, 2, 3\}$.

LOTOS brings many potential advantages: a high level of abstraction, structuring capabilities, specification simulation (execution) and verification by model-checking. There are many supporting tools for LOTOS, and in this paper, we use the CADP environment [Gar96] to reason about CSPs and TCSPs. In LOTOS, a specification is translated into a finite labeled transition system (LTS) which encodes all its possible execution sequences.

The rest of the paper is structured as follows. In the next section we show how to represent and solve a CSP using LOTOS. Section 3 is dedicated to the applicability of LOTOS to TCSPs. Concluding remarks and possible perspectives are finally presented in section 4.

2. LOTOS for CSPs

Through the example of the map coloring problem, we describe how to represent and solve a CSP using LOTOS. The problem consists of coloring each region of a given map such that no two adjacent regions have the same color. Figure 1 illustrates an example of the map coloring problem and the corresponding representation by a CSP.

The CSP of figure 1 is translated into a LOTOS specification shown in figure 2. The specification is interpreted as follows:

- Each variable, X_1 , X_2 , X_3 and X_4 , in the CSP is a variable in LOTOS.
- The domain of these variables is the sort **color**. There are three colors (*blue*, *green*, and *yellow*) which are defined as the constructor operations of the sort **color**.
- The constraints are defined in selection predicates using the two operations **not** and **and** defined in data type Boolean, **eq** defined in type **Color**.
- The domain of the next assigned variables will be reduced automatically. Only satisfied values are left in the domain.

After specifying the map coloring problem in LOTOS, we can automatically perform the following operations using the CADP toolbox:

Checking the Consistency. This consists of checking if a solution exists to this problem. In LOTOS, a problem is consistent if its corresponding specification is deadlock free (a progress is always possible). Our map-coloring problem is consistent (i.e. we can always use the three colors in the map) since the LOTOS specification is deadlock free.

Finding All Possible Solutions. In LOTOS, the simulation of the specified CSP can generate one or all possible solutions. In the map-coloring problem, finding all the solutions means all the possible ways to color the map. The simulation of the LOTOS specification leads to six solutions, such as: $\{X_1 = \text{green}, X_2 = \text{blue}, X_3 = \text{blue}, X_4 = \text{yellow}\}$, $\{X_1 = \text{yellow}, X_2 = \text{green}, X_3 = \text{green}, X_4 = \text{blue}\}$, $\{X_1 = \text{blue}, X_2 = \text{green}, X_3 = \text{green}, X_4 = \text{yellow}\} \dots$ etc.

Checking if a Path is Solution. This consists of checking if a given assignment of values to variables is consistent. For instance, is it possible to color the map with the following path: $\{X_1 = \text{blue}, X_2 = \text{green}, X_3 = \text{yellow and } X_4 = \text{green}\}$? In LOTOS, we generate first the transition systems of both path and LOTOS specification, called respectively LTS_p and LTS_s . Then with the model-checker, we verify if LST_p is a sequence of LTS_s . For our map-coloring problem, LTS_p is not a sequence of LTS_s , i.e. the above path cannot be a solution.

Completing a Partial Solution. This consists of extending a partial solution to a complete one. For instance, if $X_1 = \text{green}$, $X_2 = \text{blue}$, what are the possible colors of X_3 and X_4 ? In LOTOS, we create the partial sequence given as $X_1! \text{ green } X_2! \text{ bleue} < \text{any} > *$. Then, the model-checker completes the partial path in an incremental way if it is possible, such as $X_3 = \text{bleue}, X_4 = \text{yellow}$. Let us consider another partial path given as $X_1! \text{ green } X_2! \text{ bleue } X_3! \text{ yellow} < \text{any} > *$, the model-checking leads to a deadlock since the above path cannot be a solution.

3. LOTOS for TCSPs

Let us consider the following temporal reasoning problem:

“John, Mike and Lisa work for a company in Calgary. It takes John 20 minutes, Mike 25 minutes and Lisa 30 minutes to get to work. Every day, John left home between 7:20 and 7:26. Mike arrives at work between 7:55 and 8:00 and Lisa arrives between 7:50 and 8:00. We also know that

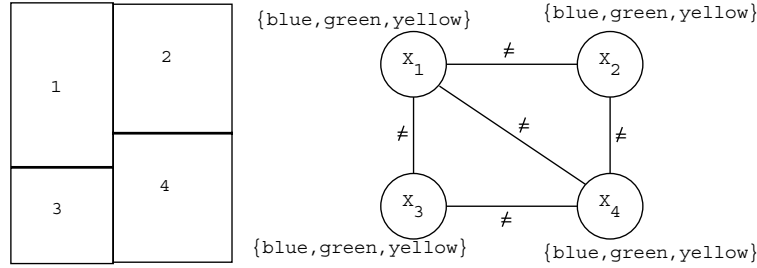


Figure 1. Map-coloring problem and its CSP representation.

```

specification Map-Coloring[Input]:exit
library Boolean endlib
type Color is Boolean
sorts color
opns
  blue (*!constructor*): -> color
  green (*!constructor*): -> color
  yellow (*!constructor*): -> color
  _ eq _: color,color->boolean
eqns forall x,y:color
ofsort boolean
  x eq x = true;  x eq y = false;
endtype
behaviour
  Input?X1,X2,X3,X4:color [(not(c1 eq c2) and not(c2 eq c4))
    and (not(c1 eq c4) and not(c1 eq c3))) and not(c3 eq c4)];
  exit
endspec

```

Figure 2. Map-Coloring Specification in LOTOS

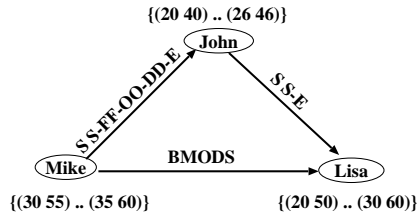


Figure 3. A Temporal Constraint Satisfaction Problem.

John and Mike meet at a traffic light on their way to work, Mike arrives to work before Lisa and Lisa and John go to work at the same time”.

Using our modeling framework TemPro[MCH98], the problem above is translated to the TCSP represented by the graph in figure 3. The nodes of the graph correspond to the three events of our story, namely: John, Mike and Lisa are going to work. The domains of the three events are the possible time intervals each event can take. Arcs are labeled with the disjunctive relations between events (disjunctions of basic Allen relations). For example, the relation $S \vee S^- \vee E$ (denoted S S- E in the graph) between John's and Lisa's

events indicates the fact that the start times of John and Lisa are equal (see table 1 for the definition of the basic Allen's relations).

This TCSP is then translated into a LOTOS specification (see figure 4) as follows:

- Each interval variable in TCSP is a variable of sort **interval** in LOTOS.
- A set of constraints between two interval variables in TCSP is a variable of sort **relation** in LOTOS.
- The constructor operation of the sort **interval** is **inv**(*s*, *e*) where *s* and *e* are natural numbers representing the **starting** and **ending** point of an interval.
- The operation **duration** of the sort **interval** computes the duration of an interval.
- The constructors of the sort **relation** corresponds to the thirteen basic relations of Allen illustrated in table 1.
- The operation **checkrel** of the sort **relation** checks the consistency of the binary interval relations.

Once we have the LOTOS specification corresponding to the temporal problem, we can reason about the story and answers the following queries:

Is the Story Consistent? In the case of TCSPs, a solution is an assignment of numeric interval to tempo-

```

specification TCSP[Mike, John, Lisa, R]: exit
library Boolean, Natural endlib
type interval is Natural, Boolean
sorts interval
opns inv(*!constructor*):natural,natural -> interval
      duration: interval -> natural
      start: interval -> natural
      end: interval -> natural
eqns forall e s, d: natural
      ofsort natural
      e >= s => duration(inv(s, e)) = e - s;
      e < s => duration(inv(s, e)) = 0;
      start(inv(s, e)) = s;
      end(inv(s, e)) = e;
endtype

type Relation is interval, boolean, natural
sorts relation
opns b(*!constructor*), s(*!constructor*), f(*!constructor*), o(*!constructor*),
      d(*!constructor*), m(*!constructor*), eq(*!constructor*), bi(*!constructor*),
      si(*!constructor*), fi(*!constructor*), oi(*!constructor*), di(*!constructor*),
      mi(*!constructor*): -> relation
      eq, neq: relation, relation -> boolean
      checkrel: interval, relation, interval -> boolean
eqns forall x, y: interval, x1, x2, y1, y2: natural, r1, r2: relation
      ofsort boolean
      eq(r1, r1) = true;
      eq(r1, r2) = false;
      neq(r1, r2) = not(eq(r1, r2));
      (x2 > y1) => rel (inv (x1, y1), b, inv (x2, y2)) = true;
      (x2 <= y1) => rel (inv (x1, y1), b, inv (x2, y2)) = false;
      ((x1 == x2) and (y1 < y2)) => rel (inv (x1, y1), s, inv (x2, y2)) = true;
      ((x1 <> x2) or (y1 >= y2)) => rel (inv (x1, y1), s, inv (x2, y2)) = false;
      ((y1 == y2) and (x1 > x2)) => rel (inv (x1, y1), f, inv (x2, y2)) = true;
      ((y1 <> y2) or (x1 <= x2)) => rel (inv (x1, y1), f, inv (x2, y2)) = false;
      ((x1 < x2) and ((y1 < y2) and (x2 < y1))) => rel (inv (x1, y1), o, inv (x2, y2)) = true;
      ((x1 >= x2) or ((y1 >= y2) or (x2 >= y1))) => rel (inv (x1, y1), o, inv (x2, y2)) = false;
      ((x1 > x2) and (y1 < y2)) => rel (inv (x1, y1), d, inv (x2, y2)) = true;
      ((x1 <= x2) or (y1 >= y2)) => rel (inv (x1, y1), d, inv (x2, y2)) = false;
      (x2 == y1) => rel (inv (x1, y1), m, inv (x2, y2)) = true;
      (x2 <> y1) => rel (inv (x1, y1), m, inv (x2, y2)) = false;
      ((x1 == x2) and (y1 == y2)) => rel (inv (x1, y1), eq, inv (x2, y2)) = true;
      ((x1 <> x2) or (y1 <> y2)) => rel (inv (x1, y1), eq, inv (x2, y2)) = false;
      rel (x, bi, y) = rel (y, b, x); rel (x, si, y) = rel (y, s, x); rel (x, fi, y) = rel (y, f, x);
      rel (x, oi, y) = rel (y, o, x); rel (x, di, y) = rel (y, d, x); rel (x, mi, y) = rel (y, m, x);
endtype

behaviour
Mike?mike: interval[(dur(mike) eq 25) and ((start(mike) >= 30) and (end(mike) <= 60))];
John?john: interval[(dur(john) eq 20) and ((start(john) >= 20) and (end(john) <= 46))];
Lisa?liza: interval[(dur(liza) eq 30) and ((start(liza) >= 20) and (end(liza) <= 60))];
R?m_j, j_l, l_m: relation
[ ((neq(m_j, b) and neq(m_j, bi)) and (neq(m_j, m) and neq(m_j, mi))
  and checkrel(mike, m_j, john)) and ((eq(j_l, s) or eq(j_l, si) and checkrel(john, j_l, liza))
  and ((eq(l_m, b) or eq(l_m, m)) or (eq(l_m, o) or (eq(l_m, d) or eq(l_m, s))))
  and checkrel(mike, l_m, liza) ) ];
exit
endspec

```

Figure 4. TCSP Specification in LOTOS

ral variables such that all the symbolic temporal relations are satisfied. Since the TCSP specification is deadlock free, that means a solution exists for this problem. Indeed, the specification simulation generates only one solution: $\{Mike!inv(30, 55) John!inv(26, 46) Lisa!inv(26, 56) R!OI!SI!D\}$. We also note that the simulation not only finds the possible interval times of events of John, Mike and Lisa but also gives the satisfied binary relations between the events. Indeed, from the generated path above $R!OI!SI!D$ represents the TCSP relations: $Mike O \sim John$, $John S Lisa$ and $Lisa D \sim Mike$.

What are the Possible Arrival Times of Lisa? This consists of looking for all possible solutions to the problem and get, for each solution provided, the end time of Lisa's event. First, we generate the LTS of

the LOTOS specification and then by using model checking, we verify if there is a path leading to a successful termination. For this problem, there is only one path: $\{Mike!inv(30, 55) John!inv(26, 46) Lisa!inv(26, 56) R!OI!SI!D\}$. This solution means that there is only one possible arrival time of Lisa, and which is 7:56.

What is the Earliest Start Time of Mike? This can be obtained by choosing an order of assigning temporal intervals to temporal events when looking for a possible solution. In this case, the order should be from left to right (from the smallest value of begin time of Mike's event to the largest value). In LOTOS, we use the model checking in the same way as the question above but we look at Mike's event. The possible start time of Mike is a set of the start times of Mike's event,

Relation	Symbol	Inverse	Meaning
X precedes Y	P	P-	$\underline{\quad X \quad} \quad \underline{\quad Y \quad}$
X equals Y	E	E	$\underline{\quad X \quad}$ $\underline{\quad Y \quad}$
X meets Y	M	M-	$\underline{\quad X \quad} \quad \underline{\quad Y \quad}$
X overlaps Y	O	O-	$\underline{\quad X \quad} \quad \underline{\quad Y \quad}$
X during Y	D	D-	$\underline{\quad X \quad} \quad \underline{\quad Y \quad}$
X starts Y	S	S-	$\underline{\quad X \quad} \quad \underline{\quad Y \quad}$
X finishes Y	F	F-	$\underline{\quad Y \quad} \quad \underline{\quad X \quad}$

Table 1. Allen Primitives

and the earliest start time is the minimum value of this set. Consequently, 7:30 is the earliest start time of Mike.

Checking if a Possible Scenario is a Solution. A scenario corresponds to a possible assignment of temporal intervals to temporal events. A given scenario is a solution to the problem if it satisfies all the symbolic temporal constraints. For example, the scenario $\{John = (20, 40), Mike = (30, 55), Lisa = (20, 50)\}$ is not a solution since it violates the constraint between Mike and Lisa. In LOTOS, we first generate the transition systems of both the path given as : $Mike!inv(30,55)$ $John!inv(20,40)$ $Lisa!inv(20,50)<any>^*$ and LOTOS specification, called respectively LTS_p and LTS_s . Then with the model-checker, we verify if LTS_p is a sequence of LTS_s . This is not the case, i.e. the above path cannot be a solution.

4. Conclusion

In this paper, using the specification language LOTOS, we have seen how to represent and solve constraint satisfaction problems in general as well as those involving temporal constraints. Through the model-checking we can for instance verify if a problem is consistent, check if a given scenario is a solution or extend a partial solution to a complete one. On the other hand, the specification simulation can enumerate all the possible solutions of a problem. Our future work is to compare the efficiency in running time and memory cost of the C code automatically generated from LOTOS specifications with the CSP algorithms based on backtrack search and constraint propagation.

References

- [All83] J.F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, 1983.
- [BB89] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. in *P.H.J. van Eijkand, C.A. Vissers and M. Diaz, eds., The Formal Description Technique LOTOS (North-Holland, Amsterdam)*, pages 303–326, 1989.
- [DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
- [Gar96] H. Garavel. An Overview of the EUCALYP-TUS Toolbox. In *Proc. of COST247, International workshop and Applied Formal Methods in System Design, University of Maribor, Slovenia, June, 1996*.
- [HE80] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.
- [ISO87] ISO. *ISO LOTOS - A Formal Description Technique Based on The Temporal Ordering of Observational Behaviour*. International Organization for Standardization- Information Processing Systems Open Systems Interconnection, Geneva, July 1987.
- [Kum92] V. Kumar. Algorithms for Constraint Satisfaction Problems: A survey. *AI Magazine*, 1992.
- [Mac77] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
- [MCH98] M. Mouhoub, F. Charpillet, and J.P. Haton. Experimental Analysis of Numeric and Symbolic Constraint Satisfaction Techniques for Temporal Reasoning. *Constraints: An International Journal*, 2:151–164, Kluwer Academic Publishers, 1998.
- [Mei96] I. Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87:343–385, 1996.
- [vB92] P. van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58:297–326, 1992.

Handling unanticipated requirements change with aspects

Ana Moreira and João Araújo

Departamento de Informática, Faculdade de Ciências e Tecnologia

Universidade Nova de Lisboa, Quinta da Torre,

2829-516 Caparica,

PORTUGAL

{amm, ja}@di.fct.unl.pt

Abstract. Supporting software evolution and maintenance are two of the major issues of aspect-oriented software development. This paper adds to aspect-orientation by proposing (i) a classification of concerns, (ii) volatile concerns to be kept separately and handled as candidate aspects independently of the crosscutting property, (iii) the use of adaptable use cases and activity diagrams to cope with generic model elements that facilitate the composition of concerns, (iv) the extension of the use case model to support our ideas.

1. Introduction

Aspect-oriented software development aims at handling crosscutting concerns by proposing means to their systematic identification, modularisation and composition. Crosscutting concerns are properties whose implementation is scattered among several implementation modules, producing tangled systems that are hard to understand, difficult to maintain and hard to evolve. Our work is at the requirements engineering level and uses facilities available in aspect-orientation to increase support for unanticipated changes in requirements.

Use cases, proposed by Jacobson [4] and later adopted by the UML [9], are a simple technique to structure the requirements of a system and to facilitate the communication with the stakeholders. However, use cases are only used to define functional requirements, leaving out global properties (such as response time, availability and compatibility) that affect the whole or part of the system. Furthermore, the crosscutting nature of some requirements (functional and non-functional) is not handled properly, even if the `<<include>>` and `<<extend>>` relationships are a good starting point [5].

This paper has two main goals: to extend the use case model so that concerns that were not modularised (e.g. non-functional requirements) can now be handled separately; to promote software evolution by externalising at this early stage volatile concerns that can be handled as candidate aspects. By externalising volatile concerns, such as business rules, that need to change on client's or market demands, we can build a stepping stone for further

management of unanticipated requirements change. In order to accomplish this we classify each concern, we keep volatile concerns separated and handle them as candidate aspects, we use adaptable use cases and activity diagrams to address generic model elements that facilitate the composition of concerns and, finally we extend the use case model to support our approach.

In the remaining of this paper section 2 introduces our approach, section 3 applies the approach to an example, section 4 discusses related work and section 5 draws some conclusions, pointing directions for further investigation.

2. Aspects to support requirements evolution

Figure 1 proposes a simple model to improve modularisation of a use case driven approach with consequent increase of software reuse and evolution.

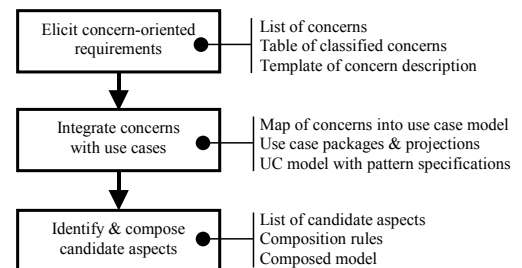


Figure 1. A model to promote requirements evolution

Task 1: Elicit concern-oriented requirements. The goal here is to produce a requirements document organized in terms of the major concerns¹ that define the problem domain. Each concern refers to a feature that the future system needs to address to satisfy the stakeholders' needs.

After identifying and defining concerns we classify them according to its type that depends on two factors: (i) longevity, that can be *enduring* or *volatile* and (ii) conceptual nature, that can be *services* or *constraints*. Enduring concerns are “relatively stable requirements which derive from the core activity of the organization and which relate

¹ A concern may be defined as a set of related requirements.

directly to the domain of the system” [8]. Volatile concerns “are likely to change during the system development or after the system has been put into operation” [8]. Constraints are properties that the system must satisfy. Services reflect functionalities that the system must offer. This information is collected in a bi-dimensional table where each cell contains the list of concerns that satisfy a combination of those two factors. Concerns are then described in more detail in a template such as the one depicted in Table 1.

Table 1. Template describing a concern

Concern #	<Concern identification >	
Name	<Concern name >	
Interrelationships	<List of concerns that this concern relates to>	
List of pre-conditions		
Pre-condition #	< What we expect is already done>	
List of responsibilities		
Responsibility #	<Responsibility name>	<Required concern>

The row “Interrelationships” lists the concerns that a given concern relates to. A responsibility is an obligation to perform a task, or know certain information [11]. A required concern is a subset of the interrelationships’ list.

Task 2: Integrate concerns with use cases. This task starts by mapping concerns into use cases (one to one), stakeholders into actors (stakeholders that directly use systems’ services are mapped into actors) and interrelationships (listed in each template) into relationships between use cases. A one-to-one mapping between concerns and use cases promotes, as we will show in Section 3, the early externalization of constraints and volatile business rules that would, otherwise, be spread throughout the original use cases, making the evolution harder.

There are six kinds of relationships between concerns. Three of them are those used by the use case model [4]: <<include>>, <<extend>> and <<inherit>>. The remaining three are new ones: <<collaborate>>, <<damage>> and <<constrain>>. While <<collaborate>> reflects a positive contribution of one concern to another, <<damage>> reflects a negative contribution. These two relationships are specific for global properties, i.e. non-functional concerns, and can be validated using the NFR catalogue [1]. Finally, <<constrain>> says that a global property restricts another concern.

The second step of this task, is concerned with handling complexity. As one concern is mapped into one use case, the resulting use case diagram is a too large diagram, even for a not so big system. One way of managing this problem is to project each global property on a use case diagram. The result of such a projection is a use case diagram with the global property connected with a

<<constrain>> relationship with all the use cases that must satisfy it.

The last step has two main goals: (1) promote concrete use cases to generic use cases; (2) describe each use case using an activity diagram with generic elements. Generic use cases can be defined in an abstract way and later be instantiated to a particular situation. A generic use case will be represented by a Use Case Pattern Specification (UCPS). Each element in a pattern specification is a role that is a UML meta-class specialized by additional properties that any element fulfilling the role must possess. Role names are preceded by “|”. (An UCPS is based on the idea of Pattern Specification [3].)

Use case roles are concerns that are more likely to change over time, such as constraints and volatile services. The idea is that such concerns, in the end, will be instantiated differently for particular configurations of a system. We can apply the same idea to relationships and have relationship roles that can be later instantiated. The instantiation is given by a rule of the form:

```
<step #.> Replace |<modelElement A>
           with <modelElement B>
```

Use cases can be described in more detail using activity diagrams. We propose a generalization of the activity diagrams to include element roles. We call these Activity Pattern Specifications (APSs). Use case roles, therefore, should be described using APSs.

Task 3: Identify and compose candidate aspects. Candidate aspects² handle crosscutting concerns and promote software evolution by externalising volatile concerns that are typically conditions, business rules and constraints. A crosscutting concern is one that is required by several concerns. (Or, it is a use case that is related to more than one use case.) However, we propose that all constraints and volatile services should be considered as good candidate aspects, independently of being crosscutting or not.

Composing use cases with candidate aspects gives the developer a possibility to understand the full picture. Composition rules are defined to weave service use cases with both constraint and volatile use cases specified with activity diagrams or APSs. A composition rule consists of a set of instantiation steps, where APS elements are replaced with concrete elements or other APS elements. It takes the form:

```
Compose <use case A> with <use case B>
{<step #.> Replace |<modelElement A>
           with
               <modelElement B>
           []
           |<modelElement B>
}
```

² The term “candidate aspect” was defined in [7].

Where “[]” represents the choice operator and “[]” denotes that the model element is a role.

3. A subway system example

This section illustrates the approach described in the previous section by means of an example based on the Washington subway system, described below:

“To use the subway, a client uses a card that must have been credited with some amount of money. A card is bought and credited in buying machines available in subway stations. The card is used in an entering machine to initiate a trip. When the destination is reached, the card is used in an exit machine that debits it with an amount that depends on the zones travelled. If the card has not enough credit the gate will not open. During periods of low usage (e.g., weekends), special package promotions are offered.”

3.1 Elicit concern-oriented requirements

List concerns. By analysing the requirements described above, we discovered concerns C1-C7 listed in Table 2. Additionally response-time, availability and multi-access are intrinsic properties in this type of systems. In particular, response time is needed as the system needs to react in a short amount of time to avoid delaying passengers; availability is needed as the system must be available when the subway is open; multi-access is needed so that several passengers can use the system concurrently; Concerns C8-C10 express these properties as concerns.

Table 2. List of concerns for the subway system

Concern #	Concern description
C1	A client buys a card in a buying machine
C2	A client must own a valid card
C3	Clients credit cards with minimum amounts of money in buying machines
C4	A client enters a subway station using a card in an entry machine
C5	A client leaves the subway station using his card in an exit machine that debits it according to the zones travelled
C6	An exit machine will not open its gate for cards without enough balance to pay the trip
C7	Special package promotions are offered during periods of low usage
C8	The system is used for several passengers simultaneously
C9	The system needs to react in time to avoid delaying passengers while they are entering or leaving the subway, or crediting their cards
C10	The system must be available for use

The number of concerns identified depends on the level of granularity used to look at the system. For example, instead of C1-C7 we could have one concern to handle each machine (entry machine, exit machine and buying machine).

Classify concerns. Each concern in Table 2 is classified according to characteristics defined in Table 3.

Table 3. Concerns classification of the system

	Enduring	Volatile
Services	C1, C3, C4, C5	C7
Constraints	C2, C8, C9, C10	C3, C5, C6

Constraints impose conditions on services. For example, constraint C6 is a pre-condition on service C5; C2 is a pre-condition on services C3, C4 and C5. Constraint C9 is a global property that C1-C7 must satisfy.

Note that C3 and C5 appear in two cells of the table, being classified both as enduring services and volatile constraints. This means that they should be divided into two separate concerns. From C3 we can derive:

- **C3A** Clients credit cards in buying machines
- **C3B** Cards are credited with a minimum amount

Similarly, for C5 we have:

- **C5A** A client leaves the subway station using his card in an exit machine that debits it
- **C5B** Exit machines calculate amount to debit cards according to the zones travelled

C3A and C5A are enduring services and C3B and C5B are volatile constraints. For example, C5B is volatile because we can change the way prices are calculated (e.g. fixed prices).

Describe concerns. Each concern is described using the template presented in Section 2, Task 1 (see Table 4).

Table 4. Template for “Exit subway”

Concern #	C5A	
Concern name	Exit subway	
Interrelationships	C2, C5B, C7, C8-C10	
List of pre-conditions		
Pre-condition 1	Card is valid	
List of responsibilities		
R1	Calculate amount to be paid	C5B, C7
R2	Check balance	C6
R3	Debit card	
R4	Register end of trip	
R5	Let client leave	

Notice that while some concerns from the interrelationships list are required by particular responsibilities, C8-C10 affect the whole concern. Also, precondition 1 is a C2 responsibility.

3.2 Integrate concerns with use cases

Map concerns into a use case model. Figure 2 depicts the use case model. Each concern was mapped into one use case. The stakeholders Client and ClientCard were mapped onto actors. Some of the relationships between use cases were identified based on the interrelationship list of each concern. Examples of each type of relationship are: <<include>> is used to relate ValidateCard with CreditCard, EnterSubway and ExitSubway; <<extend>> is used to extend the original use case CalculateAmount with Promotion. <<constrain>> is defined between C8-C10 and all the other use cases; C8-C10 are related between them with <<collaborate>> and <<damage>> relationships (this information was taken from the catalogue offered by the NFR framework [1]). We could, complement the template of the concern with this information by adding the name of the relationship between brackets after the name of the concern in the interrelationship row.

Handling complexity. In complex situations where different global properties affect different subsets of use cases, we use the projections as described in Task 2 of Section 2. Figure 2 illustrates this for Availability.

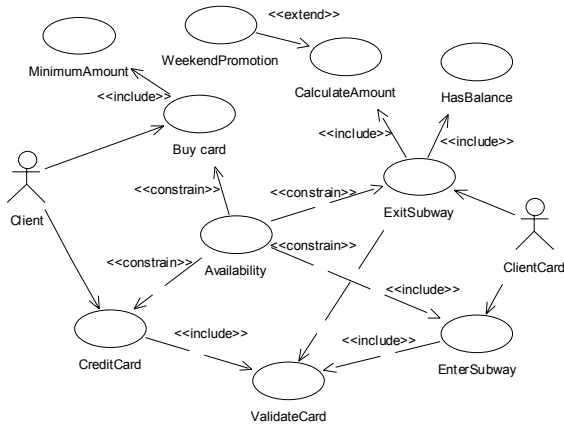


Figure 2. Projecting Availability on a set of use cases

Describe use cases with pattern specifications. A use case model is made generic by “marking” the use cases that are more likely to change as use case roles. For example, “Promotion” is a good example of a use case role, as special prices can be defined for weekends, bank holidays, seasonal periods, handicap users. It should then be preceded with “[” in the use case model. The resulting use case model is called UCPS. Later, during composition, “[Promotion” can be instantiated to “Weekend Promotion”, for example, through the rule:

Replace |Promotion **with** WeekendPromotion

Figure 3 shows APSs for ExitSubway and ValidateCard.

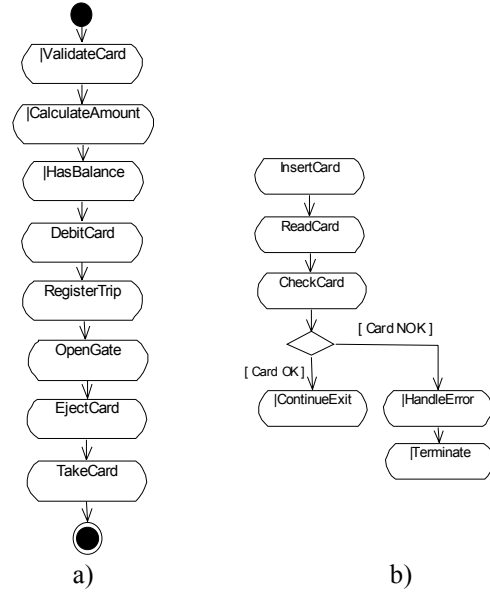


Figure 3. a) Exit subway; b) Validate Card

3.3 Identify and compose candidate aspects

List candidate aspects. The use cases ResponseTime, Multi-access, Availability, CalculateAmount and ValidateCard are crosscutting and so, are candidate aspects.

One of the contributions of this paper is that candidate aspects do not have to be crosscutting. Examples are Promotion, HasBalance and MinimumAmount. Since these are volatile business rules that can change according to the market needs, modularise them and handle them as candidate aspects promotes software evolution, since aspects are more easily enabled and disabled from a system than a class, for example.

Define composition rules. All concerns need to be composed so that the developer gets a full picture of the system. Composition will be accomplished by replacing role elements of one APS with concrete or role elements from another model.

Below there is a simple composition rule for ExitSubway and ValidateCard:

Compose ExitSubway **with** ValidateCard
 1. **Replace** |ValidateCard **with** InsertCard
 2. **Replace** |ContinueExit **with** |CalculateAmount
 3. **Replace** |Terminate **with** EjectCard

The APS resulting from this composition is illustrated in Figure 4.

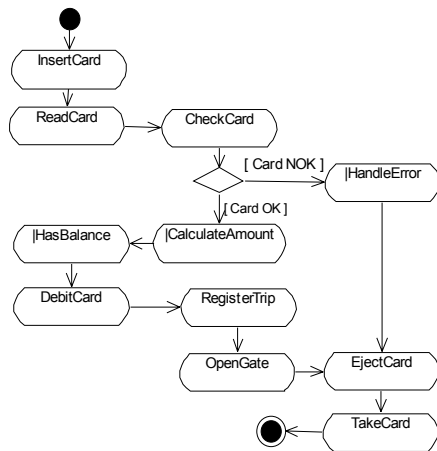


Figure 4. Diagram resulting from (partial) composition

To obtain the full view of ExitSubway, we need to compose it with all the use cases that are in any way related to this. This information is in the interrelationships row of its template description.

4. Related work

An aspect-oriented requirements model was proposed in [11]. Jacobson [5] agrees that use case extensions are a way to handle aspects during requirements and uses SDL to demonstrate that. However, his work is not on the aspect-oriented software development context and gives no systematic process to handle evolution through constraints and volatile services.

Our pattern specifications are based on the work of [3, 10]. [3] defines an aspect through role models to be composed into UML diagrams. However, the approach only handles role models and does not allow concrete elements in those models, which decreases the amount of instantiations required. Concrete modeling elements are discussed in [10] for sequence diagrams. Our only similarity with this work is the structure of the composition rules and the use of concrete elements in the APSs.

Clarke and Walker [2] define aspects using UML templates, but at design level. Also, they are concerned with how to specify the aspects rather than composing aspects with non-aspectual models.

In [6] an extension of use case modelling to handle evolution through coordination contracts is proposed. The work we present here differs not only on the level of abstraction and the use of aspects, but also the focus on concerns and composition rules.

5. Conclusions and Future Work

This paper complements our previous results in the area of aspect-oriented requirements engineering with four main innovations: a classification of concerns into services and constraints, and each one into enduring and volatile; an extension of pattern specification for activity diagrams to define role elements in a model that can be later instantiated; the externalization, i.e. modularisation, of constraints and volatile services that reflect business rules that are important in the organization; the integration of the notions above with use cases, in the context of aspect-oriented requirements engineering.

For future work we will investigate how (1) to handle possible conflicts resulting from composing APSs of concerns that have a <<damage>> relationship between them; (2) to address conflicting emergent behavior that may appear when two or more candidate aspects are allowed to co-exist; (3) to extend this approach to the modeling and design activities; (4) to develop a tool that supports the identification of concerns, their specification and composition.

References

- [1] L. Chung, B. Nixon, E. Yu, J. Mylopoulos, Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers, 2000.
- [2] S. Clarke and R. J. Walker, "Composition Patterns: An Approach to Designing Reusable Aspects", 23rd International Conference on Software Engineering (ICSE), 2001.
- [3] G. Georg, I. Ray, R. France, "Using Aspects to Design a Secure System", 8th IEEE International Conference on Engineering of Complex Computer Systems, Maryland, USA, December, 2002.
- [4] I. Jacobson, Object-Oriented Software Engineering – a Use Case Driven Approach, Addison-Wesley, Reading Massachusetts, 1992.
- [5] I. Jacobson, "Aspects: the Missing Link", Software Development, November 2003.
- [6] A. Moreira, J. Fiadeiro, L. Andrade, "Evolving Requirements through Coordination Contracts", 15th International Conference CAiSE'2003, Springer-Verlag, pp. 633-646.
- [7] A. Rashid, A. Moreira, J. Araújo, "Modularisation and Composition of Aspectual Requirements", 2nd International Conference on Aspect-Oriented Software Development, ACM Press, 2003, pp. 11-20.
- [8] I. Sommerville, Software Engineering, Addison-Wesley, 6th edition, 2001.
- [9] Unified Modeling Language Specification, version 2.0, January 2004, in OMG, <http://www.omg.org>
- [10] J. Whittle, J. Araújo, "Scenario Modeling with Aspects", IEE Proceedings Software. Under review.
- [11] R. Wirfs-Brock, B. Wilkerson, L. Wiener, Designing Object-Oriented Software, Prentice-Hall, 1999.

Integrating Security Administration into Software Architectures Design *

Huiqun Yu, Xudong He, Yi Deng, Lian Mo
School of Computer Science
Florida International University
Miami, FL 33199, USA
{yhq|hex|deng|lmo01}@cs.fiu.edu

Abstract

Software architecture plays a central role in developing software systems that satisfy functionality and security requirements. However, little has been done to integrate system design with security enforcement, which would otherwise benefits both development process and system's quality of service (QoS). This paper proposes a formal method to integrate security administration into software architecture design. We use the Software Architecture Model (SAM), a general software architecture model combining Petri nets and temporal logic, as the underlying formalism. Several techniques for designing functionality of software architectures are presented. Security modeling and administration methods are proposed. As such, SAM serves as a common platform for modeling, design and analysis of secure software architectures.

Keywords: *Software architecture, security, formal method, design, analysis*

1. Introduction

Software security is a critical concern for modern information enterprises. Breach of software security could cause a loss of money or even disaster. Software architecture plays a central role in developing software systems that satisfy functionality and security requirements [12]. Two major elements of architectures are components and connectors. Important security concerns, such as authentication and access control, arise out of interactions between components. However, architecture descriptions are typically expressed informally and accompanied by box-and-line drawings indicating the global organization of computational entities and interaction among them [1]. While informal description

of software architecture may provide useful documentation, it is impossible to analyze an architecture for consistency or determine non-trivial properties. There is no way to check that a system implementation is faithful to its architectural design.

A high degree of assurance of software security is usually achieved by independent verification of the security properties apart from good design practices and testing processes. To this end, many security policy models were proposed [11]. Various formal security verification methods were established in order to prove the correctness of security policies against the corresponding models [10, 8]. Unfortunately, security modeling and verification have been largely independent of system requirements and system design. Significant benefits can be gained by integrating system design modeling with security policy enforcement [3].

To address the above problems, we propose a formal approach to designing secure software architectures based on SAM [14]. SAM is a general software architecture model based on a dual formalism combining Petri nets and temporal logic. Security system architecture design in SAM includes two parts. One is the functionality part, which deals with the overall structure of the software architecture. The other is the security part, which handles security requirement modeling, specification, and enforcement. Several heuristics are proposed in order to guide the architectural design at both element level and composition level. Software security is enforced through well-defined rules. Analysis techniques are presented to ensure the correctness of architectural design. The main contribution of this paper is providing a formal method for integrating security administration into software architecture design on a common semantic domain.

The rest of the paper is organized as follows: Section 2 presents software architecture design techniques in SAM. Section 3 proposes security administration method. Section 4 is the conclusion.

* Supported in part by the NSF under grants HRD-0317692 and CCR-0226763, and by NASA under grant NAG 2-1440.

2. Software Architectures Design

2.1. The Structure of SAM Models

A SAM software architecture is defined by a hierarchical set of compositions, each of which consists of a set of components, a set of connectors and a set of constraints to be satisfied by the interacting components. Basically, behaviors of components and connectors are modeled by Petri nets, while their properties (or constraints) are specified by temporal logic formulas. In this paper, we use predicate transition nets (PrT nets) [4], and a linear-time temporal logic (LTL) [9]. The interfaces of components and connectors are *ports (places)*. One interface requirement is that the input and output ports of the element must be maintained at a lower level.

2.2. Element Level Design

In SAM, each element (either a component or a connector) is specified by a tuple $\langle S, B \rangle$, where S is a property specification (written in LTL), and B is a behavior model (defined by a PrT net). To define an element constraint S , we can either directly formulate the given user requirements or carry out a cause and effect analysis by viewing input ports as cause and output ports as effects. Canonical forms [9] for a variety of properties such as safety, guarantee, obligation, response, persistence and reactivity are used as guidelines to define property specifications.

The general procedure to develop B includes the following steps.

1. Use all the input and output ports as places of B ;
2. Identify a list of events directly from the user requirements or through Use Case analysis [2];
3. Represent each event with a simple PrT net;
4. Merge all the PrT nets together through shared places to obtain B ;
5. Apply the transformation technique [6] to make B more structured and/or meaningful.

2.3. Composition Level Design

SAM supports both top-down and bottom-up system development approaches. The top-down approach is used to develop a software architecture specification by decomposing a system specification into specifications of components and connectors and by refining a higher-level component into a set of related sub-components and connectors at a low

level. The bottom-up approach is used to develop a software architecture specification by composing existing specifications of components and connectors and by abstracting a set of related components and connectors into a higher-level component. Often both the top-down approach and the bottom-up approach have to be used together to develop a software architecture specification.

In SAM, only a pair consisting of a related component and connector can be composed meaningfully. Suppose that $\langle S_1, B_1 \rangle$ and $\langle S_2, B_2 \rangle$ be a pair of a related component and connector, i.e. they share some ports. The their composition is obtained through: (1) composing B_1 and B_2 by merging identical ports, and (2) composing S_1 and S_2 by conjoining $S_1 \wedge S_2$.

2.4. An Example

Consider a simple clinical information (SCI) system, which manages and maintains the personal health information. The patients' medical documents, including basic information, test results and treatment records, are classified into different access levels by the security administrator. The users (or roles), such as registration clerks, nurses, technicians, physicians etc., have different clearance levels to access those documents. The architecture of the SCI system includes four components as illustrated in Figure 1.

- the *Application System* (AS), which provides users with documents access services,
- the *Access Interface System* (AIS), which coordinates the interactions between other components,
- the *Policy Evaluator* (PE), which performs evaluation decisions based on certain security policies that govern the access to the protected resources, and
- the *Database Management System* (DBMS), which manages the medical documents.

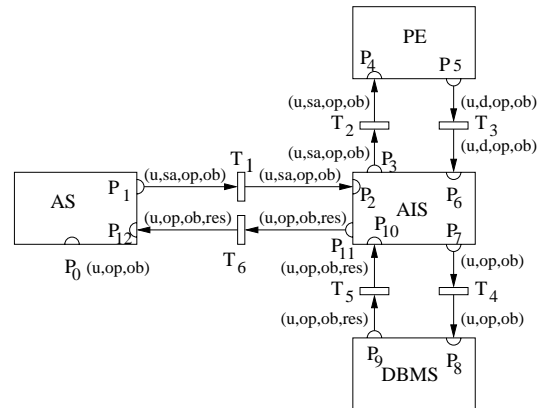


Figure 1. The SCI software architecture

Table 1. Variables in Figure 1

Variable	Description
u	User name
op	Requested operation
ob	Object name
sa	Static attributes of the user
d	Access control decision
res	Result feedback

The properties of components and connectors can be specified by LTL formulas.

- AS_S – A user request will be sent to the AIS service:
 $\forall(u, op, ob). \exists sa. (\Box(P_0(u, op, ob) \rightarrow \Diamond P_1(u, sa, op, ob)))$
- AIS_S –
 (1) AIS will invoke PE once a request is received:
 $\forall(u, sa, op, ob). (\Box(P_2(u, sa, op, ob) \rightarrow \Diamond P_3(u, sa, op, ob)))$
 (2) If AIS gets a positive decision, it will forward the user request to DBMS. Otherwise it will directly inform the user that the request was ‘denied’:
 $\forall(u, d, op, ob). (\Box((P_6(u, d, op, ob) \wedge d = 'Y'$
 $\rightarrow \Diamond P_7(u, op, ob))$
 $\wedge (P_6(u, d, op, ob) \wedge d = 'N'$
 $\rightarrow \Diamond P_{11}(u, op, ob, 'denied'))))$
- PE_S – When PE is invoked, it will return access control decision:
 $\forall(u, sa, op, ob). \exists d. (\Box(P_4(u, sa, op, ob) \rightarrow \Diamond P_5(u, d, op, ob)))$
- $DBMS_S$ – Once DBMS receives a request from AIS, DBMS will feedback a result:
 $\forall(u, op, ob). \exists res. (\Box(P_8(u, op, ob) \rightarrow \Diamond P_9(u, op, ob, res)))$

The following are connector property specifications, where every connector plays the role of a pipe.

- $$T_1: \forall(u, sa, op, ob). (\Box(P_1(u, sa, op, ob) \rightarrow \Diamond P_2(u, sa, op, ob)))$$
- $$T_2: \forall(u, sa, op, ob). (\Box(P_3(u, sa, op, ob) \rightarrow \Diamond P_4(u, sa, op, ob)))$$
- $$T_3: \forall(u, d, op, ob). (\Box(P_5(u, d, op, ob) \rightarrow \Diamond P_6(u, d, op, ob)))$$
- $$T_4: \forall(u, op, ob). (\Box(P_7(u, op, ob) \rightarrow \Diamond P_8(u, op, ob)))$$
- $$T_5: \forall(u, op, ob, res). (\Box(P_9(u, op, ob, res) \rightarrow \Diamond P_{10}(u, op, ob, res)))$$
- $$T_6: \forall(u, op, ob, res). (\Box(P_{11}(u, op, ob, res) \rightarrow \Diamond P_{12}(u, op, ob, res)))$$

The composition-level property specification (denoted by DES) is obtained by conjoining the property specifications of all components and connectors, i.e.

$$DES: AS_S \wedge AIS_S \wedge PE_S \wedge DBMS_S \\ \wedge T_1 \wedge T_2 \wedge T_3 \wedge T_4 \wedge T_5 \wedge T_6$$

One overall requirement REQ of the SCI system is that every user request must be processed, which can be specified by the following LTL formula:

$$\forall(u, op, ob). \exists res. (\Box(P_0(u, op, ob) \rightarrow \Diamond P_{12}(u, op, ob, res)))$$

One way to ensure the composition-level correctness is to show $DES \vdash REQ$. The following is a proof outline.

- (1) Assume precedence $P_0(u, op, ob)$
- (2) \forall, \exists and \Box instantiation in AS_S :
 $P_0(u, op, ob) \Rightarrow \Diamond P_1(u, sa, op, ob)$
- (3) Apply modus ponens rule to (1) and (2):
 $\Diamond P_1(u, sa, op, ob)$
- (4) Instantiate \forall and \Box in T_1 to \Diamond in (3):
 $\Diamond P_1(u, sa, op, ob) \rightarrow \Diamond \Diamond P_2(u, sa, op, ob)$
- (5) Apply modus ponens rule to (3) and (4):
 $\Diamond \Diamond P_2(u, sa, op, ob)$
- (6) Apply \Diamond absorbing rule to (5):
 $\Diamond P_2(u, sa, op, ob)$
- (7) By repeating the above Steps (4) to (6) to all subsequent element property specifications $AIS_S, T_2, PE_S, T_3, AIS_S, T_4, DBMS_S, T_5, AIS_S, T_6$, we can derive the formula in Step (8).
- (8) $\Diamond P_{12}(u, op, ob, res)$
- (9) Eliminate precedence assumption in (1) by (8):
 $P_0(u, op, ob) \rightarrow \Diamond P_{12}(u, op, ob, res)$
- (10) \Box, \exists , and \forall generalization in (9):
 $\forall(u, op, ob). \exists res. (\Box(P_0(u, op, ob) \rightarrow \Diamond P_{12}(u, op, ob, res)))$

Thus we proved $DES \vdash REQ$.

For the element-level correctness analysis, we need to show that the property specification S holds in the corresponding behavior model B . To this end, several automatic verification techniques were developed [5, 15], which include symbolic model checking, theorem proving, and reachability tree analysis.

3. Security Administration Method

3.1. Security Policy Modeling

A security policy model is a mathematical restatement of the security policy that must be enforced by the computer system. In the following, we present a framework for administration of security policies based on SAM. The administration commands is a generalization of the take-grant model [13].

Places We assume that the data types include E (Entities), Sub (Subjects), Obj (Objects), W (Rights), where $E = Sub \cup Obj$. We assume that $W = \{t, g, r, w\}$, which contains four access rights: *take*, *grant*, *read*, and *write*, respectively. Each place p represents a subject in

Sub. The type (or called inscription) of the place p is $\varphi(p) = (E \xrightarrow{m} \wp(W))$, where \wp is power set operator, and \xrightarrow{m} is mapping operator. Access matrix can be derived from the markings of places. $(e \mapsto \alpha) \in M(p)$ means that the subject p has α right(s) on the entity e under the marking M .

Transitions The state changing commands include four rules: the *take-rule*, *grant-rule*, *create-rule*, and *revoke-rule*. Each rule corresponds to a transition in a PrT net. In the following, s, s_1, s_2 denote subjects, α, β denote subsets of access rights, and e denotes an entity. We assume that firing of the transition tr update the marking from M to M' .

- *The take-rule*

The command $take(s_1, s_2, \beta)$ makes the subject s_1 to grant its β right(s) to s_2 . Formally, the command corresponds to the transition tr in Figure 2, where the dashed parts are newly added by applying the command, while the solid parts are the old ones.

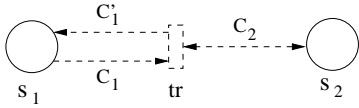


Figure 2. A PrT net for the take-rule

- $C_1 = M(s_1)$
- $C_2 = M(s_2)$
- $C'_1 = C_1 \uparrow \{(e \mapsto \alpha \cap \beta) \mid (e \mapsto \alpha) \in C_2 \wedge \alpha \cap \beta \neq \emptyset\}$, where \uparrow is the overriding operator.
- $R(tr) = Spec(M(s_1), M(s_2), \beta) \wedge (M'(s_1) = C'_1)$

Informally, $Spec(M(s_1), M(s_2), \beta)$ is a first-order formula of security policy specification¹ that stipulates the relationship on capabilities of s_1 and s_2 , as well as the set β . After firing tr , certain access rights in β of the subject s_2 are passed to the subject s_1 .

- *The grant-rule*

Using the command $grant(s_1, s_2, \beta)$, s_1 grants its β right(s) to s_2 . Its formal definition is similar to that of the take-rule, and omitted.

- *The create-rule*

The command $create(s, e, \beta)$ makes s to create an entity e and to claim β right(s) to e , whose formal definition is illustrated in Figure 3.

¹Additional elements may be needed in order to specify a particular security policy. For example, to specify Multi-level security (MLS) policy, elements such as clearance levels, and the mapping from entities to clearance levels are necessary.

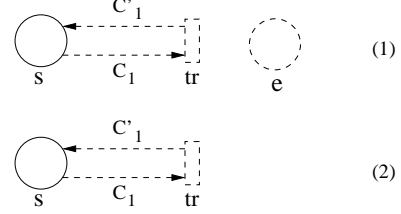


Figure 3. PrT nets for the create-rule

Case 1: if e is a subject, then a new place e and a new transition tr are added, and

- $C_1 = M(s)$
- $C'_1 = C_1 \uparrow \{(e \mapsto \beta)\}$
- $R(tr) = Spec(M(s), M(e), \beta) \wedge (M'(s) = C'_1)$.

Case 2: if e is an object, then only a new transition tr is added, and

- $C_1 = M(s)$
- $C'_1 = C_1 \uparrow \{(e \mapsto \beta)\}$
- $R(tr) = Spec(M(s), e, \beta) \wedge (M'(s) = C'_1)$.

- *The revoke-rule*

The command $revoke(s, e, \beta)$ removes β right(s) to the entity e from the subject s , whose formal definition is illustrated in Figure 4.

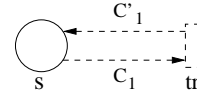


Figure 4. A PrT net for the revoke-rule

- $C_1 = M(s)$
- $C'_1 = C_1 \uparrow \{e \mapsto (\alpha - \beta) \mid (e \mapsto \alpha) \in C_1\}$
- $R(tr) = Spec(M(s), M(e), \beta) \wedge (M'(s) = C'_1)$.

3.2. Security Administration

A security policy enforcement consists of a sequence of PrT nets, such that

1. the initial PrT net contains only one place that denotes the security administrator (the super user), and
2. at each step, one of the state changing rules is applied to the current PrT net to obtain a new PrT net.

The constraint for each transition in the PrT net guarantees the correctness of the security policy enforcement.

For the SCI system, let $Sub = \{s, u_1, \dots, u_m\}$ denote the set of the security administrator and users, and $Obj = \{d_1, \dots, d_n\}$ denote the set of the patients' medical documents. The security administrator construct a PrT model as follows.

- The create-rule is used to create users and/or document. Hence, the security administrator take the access rights from all of the users.
- The revoke-rule is used to remove access rights from users.
- The take-rule and the grant-rule to exchange rights among users.

By applying these rules, a security policy model is obtained as illustrated in Figure 5. The security model plus its connections to ports P_4 and P_5 actually constitute a refinement of the *Policy Evaluator* in Figure 1.

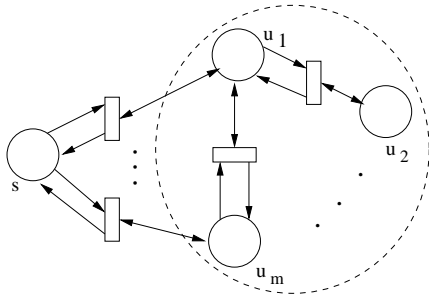


Figure 5. A security model of the SCI system

4. Concluding Remarks

This paper proposes a formal approach to designing secure software architectures. We use SAM, a general software architecture model combining Petri nets and temporal logic, as the underlying formalism. A systematic method for software architecture design and security administration proposed.

Our method has several advantages. Firstly, the method provides a rigorous way to modeling and designing secure software architectures. The method is based on a well established formalism SAM [5]. Not only SAM is capable of modeling complex software architectures, it also proves to be a powerful method for system analysis; Secondly, we integrate security administration into software architectures design, which is based on a common semantic model; Thirdly, the separation of security concerns from functionality decreases design complexity and helps to enhance system's QoS.

Interesting topics such as multi-policy enforcement, modularity in policy representation, composition, design and analysis tools are omitted in this paper. Another promising direction is aspect-oriented approach [7] to security system design, which addresses separation of concerns in software development by using specialized mechanisms to encapsulate concerns whose behavior crosscuts essential application functionality.

References

- [1] R. Allen and D. Garlan. A formal basis for architectural connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213–249, 1997.
- [2] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Longman, Inc., 1999.
- [3] P. Devanbu and S. Stubblebine. Software engineering for security: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering, ICSE'00 Special Volume*, pages 227–239. 2000.
- [4] X. He. A formal definition of hierarchical predicate transition nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets, LNCS 1091*, pages 212–229. Springer-Verlag, 1996.
- [5] X. He and Y. Deng. A framework for developing and analyzing software architecture specifications in SAM. *The Computer Journal*, 45(1):111–128, 2002.
- [6] X. He and J. Lee. A methodology for constructing predicate transition net specifications. *Software-Practice and Experience*, 21(8):845–875, 1991.
- [7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241*, pages 220–242. Springer-Verlag, 1997.
- [8] C. Ko and T. Redmond. Noninterference and intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 177–187, 2002.
- [9] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.
- [10] R. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 156–165, 2000.
- [11] P. Samarati and S. Vimercati. Access control: policies, models and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design, LNCS 2171*, pages 137–196. Springer Verlag, 2001.
- [12] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, Inc., 1996.
- [13] L. Snyder. Formal models of capability-based protection systems. *IEEE Transactions on Computers*, C-30(3):172–181, 1981.
- [14] J. Wang, X. He, and Y. Deng. Introducing software architecture specification and analysis in SAM through an example. *Information and Software Technology*, 41:451–467, 1999.
- [15] H. Yu, X. He, Y. Deng, and L. Mo. A formal method for analyzing software architecture models in SAM. In *Proceedings of COMPSAC 2002*, pages 645–652. IEEE Computer Society Press, 2002.

Learning to Select Software Components

Valerie Maxville¹, Chiou Peng Lam¹, Jocelyn Armarego²

¹*Edith Cowan University*, ²*Murdoch University*

vmaxvill@student.ecu.edu.au, c.lam@cowan.edu.au,

jocelyn@eng.murdoch.edu.au

Abstract. Developers using software components need to be confident in their selection of the most suitable component. Manual searching is time consuming and unlikely to be able to consider large numbers of components. The Context-driven Component Evaluation (CdCE) project is investigating ways to use Artificial Intelligence to assist the selection process. This paper describes our Machine Learning approach where we train a system to recognise candidates that match an ideal component specification. We utilise automated test generation techniques to create data for training the system. This results in a generic assessment system that can automatically short-list components for further investigation.

1. Introduction

Software Engineering is a movement to apply engineering principles to software development. Component-based Software Engineering (CBSE) uses software components as the building blocks for new systems, similar to hardware components. Software components are replaceable, reusable modules of executable code with well-defined interfaces [1]. As CBSE becomes more popular, we are presented with a range of components for a given application. Developers need a means for selecting the most suitable components from the growing number available in repositories and broker sites. This is not only during initial development, but also when updating components or the surrounding system.

The component selection task is normally undertaken by experts who use heuristics to determine which components are to be selected or investigated further. The desire to evaluate components using a repeatable, traceable method leads us to develop evaluation processes, such as the Context-driven Component Evaluation (CdCE) Process. Structured processes allow us to standardise how we deal with candidates, but a manual assessment is unable to scale to large numbers of components. We propose that Artificial Intelligence (AI) techniques be applied to automate parts of the selection

process to allow the consideration of larger numbers of candidates.

A common approach to assessing components is to take weighted scores against a list of attributes and aggregate them. An expert's holistic view of a candidate may incorporate interplay between attributes – conflicting or reinforcing its suitability. This interplay can be recorded as a series of relations between attributes. Rules associated with these relations can then interact with the candidate's "scores" against attributes and their overall evaluation. This interplay between attributes is lost in a numerical aggregation. In this paper we describe our approach to selecting components, which works from a specification of the ideal component, then uses machine learning and test case generation techniques to train the system to automatically evaluate candidate components. Our Selector system automates the determination of rules and building the knowledge base so that the user interface is simple and intuitive. We address the issues of scalability, attribute interplay and the ability to explain the reasoning behind a selection decision.

The following section discusses component selection, AI techniques and the application of AI to component selection. Section 3 describes our Machine Learning approach to selection. A case study is presented in Section 4. The CdCE Project is described in Section 5, with conclusions and future work in Section 6.

2. Related work

Selection of components is a similar problem to selection of Commercial Off-The-Shelf (COTS) software, and COTS research can be applied to component selection. Research in component selection begins with defining the selection criteria. Most selection approaches have a component model that describes the criteria or attributes to be used in the assessment, often implemented as a hierarchy. A discussion of these models is given in [3]. Other schemes develop a hierarchy for the specific problem [4][5]. The relative importance of the criteria may be determined using a structured approach such as AHP [2][5]. An assessment of each component against

the criteria is then carried out, most often as a manual process. A recommendation or ranking can then be determined. This normally involves an aggregation of results using the Weighted Scoring Method (WSM) or the AHP [2]. In other cases, techniques such as Outranking are used [6]. Recent research has begun to use AI techniques to address issues with assessing components, in particular the inherent problems with aggregating results. Neuro-fuzzy [7] and Rough fuzzy sets [8] have been used to deal with imprecision and uncertainty in component assessment, while overcoming some overheads of determining the original fuzzy sets. Most techniques are more applicable to in-house repositories where the documentation of components can be standardised and detailed, with up to 1320 attributes for each component [9]. Our project is concerned with third party components sourced from a range of repositories. We then have a very large number of components to screen and rudimentary information about them. This leads us to AI to carry out both coarse screening and more in-depth analysis of the technical features of candidate components. It is important that the overheads for the AI technique are low as each selection process will have new requirements and is thus a new problem.

Artificial Intelligence is a field that provides a range of techniques for representing and processing knowledge. When selecting an AI technique, it is important to consider the features that are needed, and which are more critical to the particular problem. In the component selection problem, we are trying to classify the components as being acceptable or rejected. We also want to be able to adjust thresholds to include or exclude more candidates, where criteria may have been too restrictive or lenient. Working with metadata from various sources introduces inconsistency to our data, so some tolerance for missing or uncertain data is important.

Tables 1 and 2 in this document show how traditional and hybrid AI systems perform against eight criteria, all which are quite important for the automated selection of components. Knowledge representation is important to component selection as our process is working with the metadata supplied by vendors and brokers, and the results need to be understandable to users. As we will be working with information from diverse sources, there is a risk of missing and uncertain data. Many of the selection criteria for components can be considered “a match” or “not a match”, but the facility to deal with imprecision may be more useful when looking at how well a description meets our needs, e.g. how close the cost of the component is to our ideal.

Our interest in AI is to automate the selection of components. An automated assessment is unlikely to be trusted unless there are explanation facilities to give the reasoning behind any decisions. The traditional AI systems that perform well on explanation ability rate

Table 1. Comparison of Traditional AI Techniques, adapted from [10]

Feature	Expert Systems	Fuzzy Systems	Neural Networks	Genetic Algorithms	C4.5 Classification
Knowledge representation	+	++	--	-	+
Uncertainty tolerance	+	++	++	++	-
Imprecision tolerance	--	++	++	++	--
Adaptability	--	-	++	++	++
Learning ability	--	--	++	++	++
Explanation ability	++	++	--	-	++
Knowledge discovery and data mining	--	-	++	+	+
Maintainability	--	+	++	+	++

Table 2. Comparison of Hybrid AI Techniques

Feature	Neural Expert Systems	Neuro-fuzzy Systems	Evolutionary Neural Networks	Fuzzy Evolutionary Systems
Knowledge representation	+	++	--	++
Uncertainty tolerance	++	++	++	++
Imprecision tolerance	++	++	++	++
Adaptability	++	++	++	+
Learning ability	++	++	++	+
Explanation ability	++	++	--	++
Knowledge discovery and data mining	--	-	++	+
Maintainability	++	++	++	+

poorly on adaptability, learning and maintenance. This would lead to a trade off where the reasoning can be explained, but there is a heavy load on the expert to develop and tune rules – diminishing the advantage of using AI. Neural expert or neuro-fuzzy systems may overcome this, assuming we can generate data to train the neural network. Our interest in looking for components

on the Internet does imply an interest in data mining and an AI technique that can extend to knowledge discovery would be an advantage. Although not one of the criteria in the comparison tables, we also want to be able to deal with the interplay between attributes. Any of expert systems, fuzzy systems or neural networks is capable of encoding these dependencies.

It is clear that an AI technique for component selection would ideally rate well in all of the above categories. For this investigation, we have chosen to use the C4.5 decision tree classifier [11]. C4.5 takes labelled data and grows a large tree which is pruned to create a decision tree of understandable size. As can be seen in Table 1, this approach rates well in all features except uncertainty and imprecision tolerance. We will be addressing these very important issues in future work by investigating other AI techniques for classifying data. An evaluation of the relative performance will then be carried out.

3. Intelligent Selection

Any selection process begins with a requirements specification for comparison with candidate components. We work with an ideal specification based on an XML Schema template [12]. The ideal specification includes all attributes of interest to the application developer. The specification is annotated with information regarding the priority of attributes and any interplay between them. Our system combines the ideal specification with the schema definition to create an internal model of the desired component. The system is not tied to the CdCE component model and can import any XML Schema and instance documents for a generic selection problem.

The attributes describing the components are split into four categories, according to datatype. This binding is determined from the Schema document. The simplest is “string” where there can only be a single value per candidate. An example is the `dc:creator` attribute – the Dublin Core [13] tag representing the software developer. We also have date and numeric attributes where an optimal value is specified along with optional minimum and maximum values. The final datatype is the multiString. A multiString is used in situations where an attribute can have one of a set of values. MultiString attributes are split into multiple attributes for input into the machine learning software. For example, the desired values for the `operatingSystem` attribute may be UNIX and Linux. We map these to `operatingSystem_UNIX` and `operatingSystem_Linux` for training data and the data to be assessed.

An issue in using supervised learning techniques¹ is that they require either large amounts of historical data, or a manual evaluation of input data. We have used techniques from test data generation to create a set of training data from the internal model of the ideal component. Values for each attribute are grouped into equivalence classes, and the attributes themselves are grouped according to how they influence the evaluation. The internal model provides enough information to determine whether a component is accepted or rejected, which is used to attach a result to the generated datasets.

Exhaustive generation of data is not practical. With 32 attributes in use of simple types (Boolean for this calculation), we would need over 4,000 million data entries to cover all combinations of the data. The algorithm for generating the data targets groups of entries as “lessons” to train the system to learn a specific aspect of the assessment. The lessons first focus on distinguishing datasets to help the system learn where the border between acceptance or rejection of a component lies. It also creates lessons around areas of complex interaction between attributes to reinforce the learning process. The size of the generated dataset is dependent of the number of attributes in the selection task, and the amount of interplay between them. Training the C4.5 classifier is not greatly affected by the size of the dataset, and takes a few seconds. We have used the same data with an Artificial Neural Network which takes over 20 minutes to generate the classifier.

A balance is required between the amount of data in each output category. Early training sets oversimplified the decision to “reject all” due to the selection of training data. A component selection process is likely to reject a high percentage of candidates, but using data that follows that distribution skews the training. We are continuing to investigate how to balance the training to work with an optimal size and number of lessons. The best results to date have come from generating between $\frac{1}{2}$ to $\frac{2}{3}$ of the training data falling within the acceptable class.

4. Case Study

In this case study we revisit a manual selection exercise working with real world data, updating it to use machine learning. The scenario for the case study is the selection of a software component to provide scientific calculator functionality. The XML for the ideal specification is given in Figure 1. It uses the CdCE Schema as a base

¹ Supervised learning relies on a manual labelling of the training data for the system to learn the patterns for each classification. Unsupervised learning works with the patterns formed within the training data and attempts to group them into clusters. Data falling inside a cluster can then be labelled according to the closest cluster.

with 32 attributes in the following categories: three numeric, one date, 21 String and seven multiString (enumerated). The Schema allows some tags to be repeated, which map to multiString attributes in our system. Potential component information was taken from four online sites. These had been assessed previously with a manual application of the CdCE process. A summary of that assessment is in Table 3. It gives an indication of the rejection rate and the number of possibilities that would need to be considered in a component selection problem. The automation of selection is highly dependent on the adoption of a standard specification format by component brokers.

The attributes used in the case study are shown in Table 4. The attributes are classified as being *mandatory*, *preferred* or *other*. *Mandatory* attributes must all be met for the candidate to be accepted. A threshold value modifies the number of *preferred* attributes that need to be matched to accept the candidate. The *other* attributes do not affect the assessment. This provides three equivalence classes for our data generation. Test generation uses equivalence classes to reduce the number of test cases by having one value represent the whole class of values, and may have rules for the output based on the input class. For training the system, we use the equivalence classes to enumerate the combinations of attribute values inside and between classes, and the corresponding classification for that component. In this case, three of the attributes are *mandatory* and five are *preferred*, the remaining attributes are categorised as *other*. We have arbitrarily selected a threshold of 0.5 which rounds down to two out of five preferred attributes for acceptance.

We use the Weka system [15] to access machine learning algorithms. Weka uses ARFF format files where attributes and values are listed, then the training and/or test data. For supervised learning, the last attribute denotes the classification of the entry, in this case `result=accept/reject`. As mentioned previously, the generated training data is grouped into lessons. We start with lessons in acceptable attribute values, then look at what values will lead to rejection. Parameters on the generation can adjust the number and size of lessons. The lessons focus on the patterns of attribute values that are near the border of acceptable/unacceptable. Random selection of training data would almost certainly result in all candidates being classified as rejected. Our solution is to apply Boundary Value Analysis (BVA) techniques. We select training data that sits close to the boundary between acceptance and rejection, along with some more straight-forward entries. This has prevented the classifier from over-simplifying its decision tree and allows us to work with relatively small training sets.

Table 3. Case Study Manual Assessment [14]

Site	Number of entries	Number of candidates
I	>8,000	1
II	>12,000	7
III	>36,000	4
IV	>30,000	0
Total	>86,000	9 (3 duplicates)

Table 4. Case Study Ideal Specification

Attribute	Type	Importance	Values
Description	Multi-String	Mandatory	Scientific Calculator
Development Status	String	Mandatory	Mature
Licence	String	Preferred	GPL
Price	Numeric	Preferred	\$0-\$75
Development Language	Multi-String	Preferred	Java C++
Operating System	Multi-String	Mandatory	Linux
Memory	Numeric	Preferred	5-70Mb
Disk Space	Numeric	Preferred	10-90Mb

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/xsl" href="swvML_ap_1.1.xsl"?>
<Description xmlns="http://www.scis.ecu.edu.au/research/PhD/vmaxvill/swvML/1.0/"
  xmlns:dc="http://purl.org/dc/elements/1.0/"
  xmlns:sw="http://www.scis.ecu.edu.au/research/PhD/vmaxvill/swvML/1.0/"
  about="http://www.scis.ecu.edu.au/research/PhD/vmaxvill/swvML/1.0">
  <dc:description type="mandatory">scientific</dc:description>
  <dc:description>calculator</dc:description>
  <sw:devStatus type="mandatory">mature</sw:devStatus>
  <sw:licence type="preferred">GPL</sw:licence>
  <sw:price type="preferred" min="0" max="75">25</sw:price>
  <sw:technical>
    <sw:devLanguage type="preferred">Java</sw:devLanguage>
    <sw:devLanguage>C++</sw:devLanguage>
    <sw:operatingSystem type="mandatory">Linux</sw:operatingSystem>
    <sw:systemRequirements>
      <sw:memory type="preferred" min="5" max="70">20</sw:memory>
      <sw:diskSpace type="preferred" min="10" max="90">40</sw:diskSpace>
    </sw:systemRequirements>
  </sw:technical>
</Description>
```

Figure 1. Ideal Specification in XML

Our system provides great flexibility in the generation of training data. We use Weka's implementation of the C4.5 classifier which outputs a decision tree. It also gives an analysis of the resultant tree's performance against the training and test data. The derived decision tree matched the model of the candidate selection criteria and when applied to the training data, it correctly classified 100% entries. Another test of the classifier was run against simulated data and correctly classified all the components and selected 27 out of 2000 components as potential candidates.

We then ran the trained classifier over real component data where it identified 17 suitable components for the

578 that were considered. Although the four repositories offered over 86,000 entries, we worked with a subset of those matching the search criterion “calculator” as manual conversion of all entries to XML was impractical. Incorrect results were given for less than 7% of the data, in situations where values for attributes were missing. Classification of missing data is one of the limitations of C4.5. If it has not seen a particular value for an attribute, it will still try to classify the instance according to its decision tree, with unpredictable results. In our data, missing information was replaced with “-” for text attributes and -1 or 1000 for numeric attributes. We are investigating the substitution of average or default values for missing values, as well as alternate Machine Learning approaches to improve the handling of missing data.

At this point, we can consider updating or tuning the ideal specification. Using the facilities provided by Weka, we can look at the component data as individual attributes or as groups of attributes. Statistical information about individual attributes helps us to adjust ranges for numeric values. Clustering tools help us to find components that have a similar profile to our ideal specification. We can then adjust the ideal specification, retrain the classifier and re-run the component data to get a tighter match on suitable components.

5. Context-driven Component Evaluation²

This work is part of the CdCE Project. The Project aims to develop strategies for the assessment of software components, both through static comparison of developer requirements to a candidate component specification and by generating context-driven tests for the dynamic assessment of short-listed components. We address the issues of sourcing, selection and evaluation of software components, with indirect benefits in testing and trust. The process is driven by a specification of the ideal component and its operating context, which provides a foundation for the automation of the selection process. We focus on the selection of third party components from commercial and open source brokers and repositories where the format and detail of component documentation can vary widely.

An important attribute of third party software components is that they are written for the general case. They then require contextual information and testing to fully evaluate their suitability to an application [16]. The developer needs to know that the component is not only reliable and meets its specification, but that it is suited to the target system. Component certification can improve confidence and trust, but is not sufficient reason for a particular selection as it does not take context into

account and cannot ensure that a component will behave correctly in another environment [17]. Our ideal component specification includes details of the requirements for the component and aspects of the target system to allow a context-aware evaluation of a component's suitability.

Figure 2 shows our process for component evaluation. In the first step we define the requirements which become the ideal component specification. The ideal component is specified on two levels, metadata for descriptive information, and a formal specification of the interfaces and behaviour in Z notation. Step 2 searches for candidates matching the ideal specification, resulting in a short-list for further examination. Abstract test cases are generated for the components in Step 3, based on the formal specification of the ideal component. The tests can also be used for system and regression testing. An adaptation model is developed for each candidate in Step 4 and used to adapt the abstract test cases to match each of the short-listed candidates.

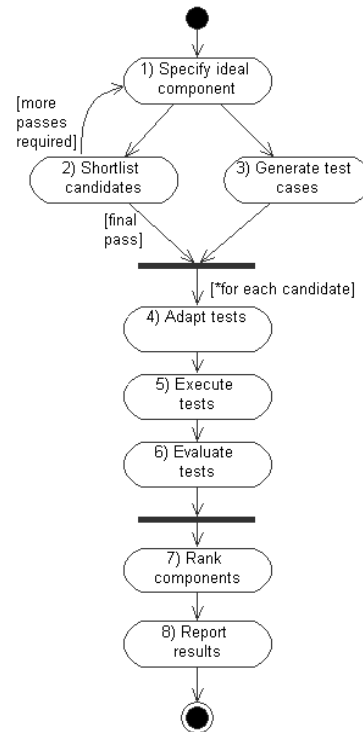


Figure 2. Activity diagram for CdCE Process

The tests are executed against the candidates in Step 5, and the test and short-listing results are combined in Step 6 to get an overall picture for each component. In Step 7 we look at the results across the candidate components to generate a comparison. This may involve aggregation for scores against criteria, or other methods such as the C4.5 classifier described in this paper. We can then move to Step 8 and provide a recommendation for component

² Formerly known as the Context-driven Component Testing (CdCT) project

selection, including reasons behind the recommendation, and information to assist in adapting and integrating the component. A more detailed description of the process appears in [18] and [14]. We are currently developing a tool to assist developers through the CdCE process, linking to classification and test generation software and compiling the results of each step for generation of the recommendation(s) in Step 8.

6. Conclusion

We have explored the use of Machine Learning algorithms for the selection of software components. Our case study results show promise, with the generated data training the C4.5 classifier and providing an appropriate decision tree. It then gave correct classification for all candidate components (in minutes) compared with a manual approach which missed some candidates and took over eight hours. Our training data generation overcomes a major issue with supervised Machine Learning in that it does not require large amounts of historical or statistical data as we generate the training data and labels (accept/reject) from a model. We also address the problems of aggregation-based component selection approaches where the relationships between components are lost.

Machine Learning is not normally economical for one-off classification problems. Each new search for a component is a new problem with different selection criteria. Our approach works from the ideal specification, which is always necessary for component selection. We automate the training data generation from the ideal specification using generic techniques and can easily train for the selection task at hand. The result is a considerable automation of the selection process requiring a small amount of expert time. We are currently applying this to the short-listing or filtering stage of component selection, but it can also be used for the more technical evaluation required later in the selection process (Step 7 of CdCE Process). The Machine Learning tools can also be used to adjust or tune the ideal specification based on statistical and clustering information.

This work is one way that AI can be applied to benefit those in the computing community. We plan to extend this work by investigating and evaluating other classifiers and Neural Networks to further utilise the generated training data for supervised learning of component selection criteria. We are also looking at improving the data representation so that more information can be fed back into the process via clustering and other learning techniques.

References

- [1] C. Szyperski, *Component software: beyond object-oriented programming*. New York: ACM Press, 1997.
- [2] T. L. Saaty, "The Analytical Hierarchy Process", McGraw-Hill, 1990
- [3] S. Sassi, L. Jilani and H. Ghezala, "COTS Characterization Model in a COTS-based Development Environment", Asia-Pacific Software Engineering Conference (APSEC), Chiang Mai, Thailand, 10-12 December, 2003
- [4] J. Kontio, "OTSO: A Systematic Process for Reusable Software Component Selection". Tech. Report UMIACS-TR-95-63, University of Maryland, 1995.
- [5] M. Ochs, D. Pfahl, G. Chrobok-Diening and Nothhelfer-Kolb, "A Method for Effective Measurement-based COTS Assessment and Selection – Method Description and Evaluation Results". Tech. Report IESE-055.00/E, Fraunhofer IESE, 2000.
- [6] M. Morisio and A. Tsoukis, "IusWare: a Methodology for the Evaluation and Selection of Software Products", IEEE Proceedings of Software Engineering, Vol. 144(3), pp. 162-174, June 1997.
- [7] Y.-H. Kuo, J.-P. Hsu and M.-F. Horng, "Neuro-fuzzy Based Search Robot for Software Components", International Journal on Artificial Intelligence Tools, Vol.8(2), pp. 119-135, 1999.
- [8] D.V. Rao and V.V.S. Sarma, "A Rough:Fuzzy Approach for Retrieval of Candidate Components for Software Reuse", Pattern Recognition Letters, 26(6), March, 2003.
- [9] S. Nakkrasae, P. Sophatsathit and W.R. Edwards, Jr, "Fuzzy Subtractive Clustering Based Indexing Approach for Software Components Classification", Proceedings of the 1st ACIS International Conference on Software Engineering Research & Applications (SERA'03), San Francisco, USA., June 25-27, 2003, pp. 100-105.
- [10] M. Negnevitsky, "Artificial Intelligence: A Guide to Intelligent Systems", Addison Wesley, 2002
- [11] J. Ross Quinlan, "C4.5: programs for machine learning", Morgan Kaufmann Publishers Inc., CA, 1993
- [12] World Wide Web Consortium. "XML Schema" [Web page]. Accessed 20/12/2003, from the WWW: <http://www.w3.org/XML/Schema>, 2003.
- [13] Dublin Core Metadata Initiative. "DCMI Website." Accessed 20/12/02, from the World Wide Web: <http://www.dublincore.org/>, 2002
- [14] V. Maxville. "Context-driven Component Testing Project Website" [web page]. Accessed 6/9/03, from the WWW: <http://www.scis.ecu.edu.au/research/PhD/vmaxvill/>, 2003
- [15] I. Witten and E. Frank, "Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations", Morgan Kaufmann Publishers, 2000.
- [16] E. Weyuker, "Testing Component-based Software: A Cautionary Tale". IEEE Software, 15(5), pp. 54-59., 1998
- [17] A. Cechich, M. Piattini, and A. Vallecillo, "Assessing Component-based Systems", In: Cechich et al. (Eds.) Component-Based Software Quality, LNCS 2693, pp. 1-20, Springer-Verlag Berlin Heidelberg 2003.
- [18] Maxville, V., Lam, C. P. and J. Armarego "Selecting Components: a Process for Context-Driven Evaluation", Asia-Pacific Software Engineering Conference (APSEC), Chiang Mai, Thailand, 10-12 December, 2003

Organizational Knowledge: an XML-based Approach to Support Knowledge Management in Distributed and Heterogeneous Environments

Carmen Maidantchik, Gleison Santos, Mariano Montoni
Federal University of Rio de Janeiro - COPPE
Caixa Postal 68511 – CEP 21945-970 – Rio de Janeiro, Brazil
Phone: +55-21-25628699 Fax: +55-21-25628676
E-mail: {lodi, gleison, mmontoni}@cos.ufrj.br

Abstract. This work presents an approach to support knowledge identification, capturing and maintenance in distributed and heterogeneous environments. A worldwide telecommunication company has been employing an XML-approach that represents diverse data through distinct files. The tags identify the data, facilitate its understanding, and structure all related information. An importation process associates the incoming data with stored procedures placed in the main repository. The insertion, update, or removal of data and related knowledge are easily performed, facilitating the management of knowledge that continuously evolves.

1. Introduction

This work presents an approach to support knowledge identification, capturing and maintenance in distributed and heterogeneous environments. A worldwide telecommunication company that supplies high technology services for all Brazilian states developed a software system to support the activation process of its clients' network. The required rationale to launch a new telecommunication service used to be an assembled of various kinds of knowledge that were previously located in corporate systems, documents, personal programs, and within working members. An effective implantation of these telecommunication services guarantees the company market position. In order to assure high quality software, the design process considered the distinctiveness of the company activation groups and posed some questions to be considered. How system requirements and knowledge could be extracted, gathered, and understood without delaying the application development? How the acquired knowledge could be represented in such a way that telecommunications specialists could understand? How this representation could be structured to allow the description of several kinds of data? As knowledge evolves, changes, and matures, how the activation process could be codified to avoid numerous modifications?

Answering the first question, user centered design methods along with knowledge acquisition techniques were used in an evolutionary software life cycle process [1, 2]. In order to achieve an efficient representation, our software engineers group constructed an XML-based organizational knowledge repository aiming at integrating all expertise that was previously distributed among different departments. As XML (eXtensible Markup Language) is a markup language, the tags identify the data and may explain details about the knowledge origin, default format, author, and so on. Therefore, XML is easily understood by human beings and efficiently represents a large variety of different kinds of knowledge. The last question is related to knowledge evolution. As the organization strategic directives may change according to market, internal processes and economical changes, corresponding support software also suffer adaptations. To avoid software maintenance each time an evolution phase occurs, a flexible importation process was designed. The goal was to import knowledge from other systems repository independently of the knowledge representation format and system technology.

In the following section we describe some basic knowledge management concepts. In section 3, we present our XML-based approach to construct knowledge repository through organizational knowledge identification, capture and maintenance. The section 4 presents future work and directions. Finally, in section 5, we conclude with the benefits of our proposal.

2. Managing Organizational Knowledge

Currently in some organizations, it is possible to observe a tendency to establish activities to acquire, organize and communicate tacit and explicit organization members' knowledge. The outcome benefit is that other members can make use of available information to enhance their

effectiveness and productivity [3, 4]. The set of these organizational activities is commonly known as Knowledge Management (KM). The main objective of KM is to make relevant knowledge accessible and reusable by organization members. However, considering the diversity of information types, it is necessary to identify the relevant ones. Knowledge that facilitates or improves organization members' activities execution and, therefore, probably results in benefits for the organization, have to be distinguished [5]. According to M. Alavi and D. Leidner [6], KM also provides the means to create innovative organizational practices to support communication and collaboration among professionals from the same or different domains. Several authors suggest a basic set of fundamental activities to systematically manage knowledge [4, 5, 7]: (i) identify important knowledge that can be used to increase its visibility and access; (ii) capture and store useful knowledge in a repository. Expertise acquisition requirements must meet defined knowledge types (i.e. explicit or tacit) and easiness to access its sources; (iii) maintain knowledge in the organizational memory through update or removal of irrelevant, useless and outdated information. The next section presents an XML-based approach to support these KM activities. Some of the benefits of this approach are also described.

3. The XML-based approach to Identify, Capture and Maintain Organizational Knowledge

Within the worldwide telecommunication company, knowledge was not always represented in an adequate format to allow its identification and capture. The management of such wisdom is very useful to support organizational managerial activities, such as network resources management. Consequently, it was imperative to develop an approach that could efficiently identify knowledge sources, capture and store the data in a repository. As a result, other systems and organization members can make use of the stored information.

Existing corporate systems provide simple text files, denominated flatfiles, as a mirror of their databases. Since there is no standard format, the generated flatfiles are different from each other. As a consequence, analyses and interpretation of data placed in the archives are laborious tasks. Besides that, flatfiles may contain redundant or incomplete data, and the documentation is inadequate and not frequently updated. Due to these problems it was mandatory to develop an approach that could provide an efficient data integration and exchange mechanism. It should also store information about the system from which the data were extracted, flatfile structures and details of each processed data item. Once the knowledge

has been identified, it has to be captured and stored in a knowledge repository, in such way that organization members could rapidly make use of it. Nonetheless, all acquired knowledge placed in the repository should constantly be reviewed and updated in order to guarantee reliability.

The knowledge representation format is an important aspect that was considered to efficiently manage the whole data. Markup languages, as XML, can be used to describe knowledge structures and to support the organizational memory development [8, 9]. XML provides a standard layout to communicate and exchange information and knowledge among different systems. This language makes possible to create multiple visions of the same item and also provides an easy mechanism to capture, store, present and recover knowledge [10]. The use of XML allows uniform knowledge acquisition, systems interoperability and offers efficient mechanisms for information recovery. Considering these benefits, we developed an XML-based approach to construct a knowledge repository that integrates data from different sources, previously represented in different formats. The mechanism also supports knowledge identification, capture and maintenance activities.

3.1. Constructing an XML-based Organizational Knowledge Repository

The construction of an organizational knowledge repository requires data sources identification and integration. The information is generated by different corporate systems and placed in non-standard flatfiles. The merging process starts with the identification of related knowledge and then, connecting the items through XML files. Due to the great amount of information, the processing time could make the integration unfeasible. Two archives were created directly from the flatfile. The first one contains all data stored in the main knowledge repository. The other archive holds only the latest modifications. By using this approach, only new, updated or removed data originated from corporate systems are processed. Therefore, the first importation of modified data corresponded to a small percentage of the total generated volume.

Through XML, it was possible to guarantee file format independence and to associate semantic information of data to their values. The XML files construction follows three steps: (i) redundancies removal, (ii) definition of information structure, and (iii) elimination of irrelevant data. The importation program converts flatfile into XML archives in only one step. These operations are performed simultaneously, reducing processing time. Figure 1

illustrates this process. Flatfiles exported from corporate systems are parsed and transformed into two XML files. The complete file contains all data that exists in the original flatfile, while the partial one contains only the data that was modified since last importation.

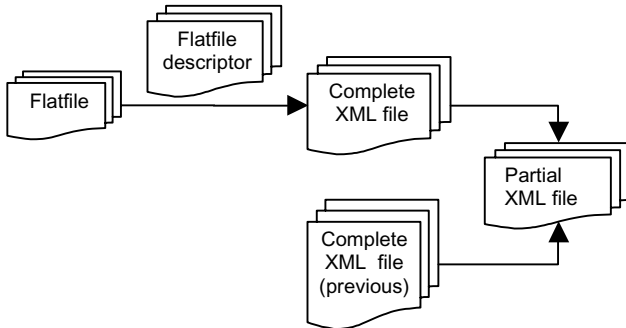


Figure 1. Conversion process from flatfile to XML archives

The partial file is generated by comparing a complete previously generated file with its correspondent current complete one. The program also inserts an index to optimize the importation process. For instance, complete XML archives containing clients' information have approximately 170 Mbytes, while partial XML archives have approximately 50 Kbytes, i.e., just 0.03% of the complete XML file size. Processing time is reduced because with this approach, it is not necessary to execute queries to verify whether a specific item has to be inserted or deleted in the knowledge repository. With XML tags, this information is already available in the archives.

The first step to construct a knowledge repository is to identify data suppliers. In our approach, another XML archive contains information about knowledge sources. A header and a body compose this file. The header section contains managerial information, such as knowledge source objectives and last modification date. The body section is formed by a set of tags holding information about XML archives to be created, knowledge format and structure, data types (e.g. clients' or circuits' information), source location, items descriptions and corresponding tags that identify them, rules, and so on. The capture and maintenance knowledge processes are facilitated since data sources descriptions are represented in XML archives. Header, body and footer sections compose the partial and complete XML archives. The header contains a description of the file and information about its creation. It also contains information about the action to be executed for each register: "D" means delete, "I" means insert, and "U" means update. The XML file body contains the knowledge to be imported, and the

footer contains information about the total number of registers in the file.

3.2. XML Importation Process

The importation process is independent of the XML files structures. An auxiliary file contains information about the stored procedures that should be executed when a certain data type has to be imported. During importation process, the XML archive is parsed to identify operations (insert, update, delete) and stored procedures that execute these operations. Within this approach, stored procedures depend only on the data type being imported. Therefore, changes in XML files structure do not imply importation program maintenance.

Missing parameters are reported by the stored procedure. The existence of exceeding attributes in the XML file do not interferes on the stored procedure execution since the importation program ignores this information. Figure 2 illustrates the importation process. An XML archive that contains the data to be imported ("Partial XML file") is parsed together with another file ("Stored procedure descriptor") that defines all stored procedure associated to the data type that will be imported into the database.

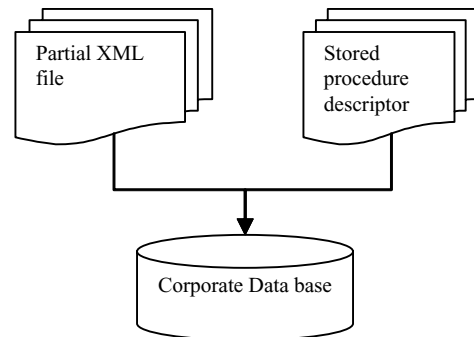


Figure 2. XML file importation process

3.3. Importing Knowledge with XML

Once knowledge has been captured, structured and stored in the repository, it can be imported into any corporate database. This process can be performed independently of knowledge representation format since XML archives map knowledge items to specific stored procedures that should be executed to import the data.

A header and a body compose the parsed XML archive to import knowledge about a network management system. The header section has information about the file, for example, its objective and last modification date. The body contains information about the stored procedure

parameters used during the importation process. The body also contains information about stored procedure name and output values, parameters, data size and type, XML tags that identify a specific parameter value in the archive to be imported, and so on.

4. Future Work

The developed system generates more than 20 different types of topologies of the telecommunication clients' network. Each type is related to one service that the company provides. Currently, the application supports the management of approximately 235,500 clients, 52,600 circuits, 18,000 network accesses (e.g.: optic fibers, radios, and satellites), and 54,700 different sites spread throughout 6 regions of the Country. For each data that is held during the activation process of a telecommunication service, several diverse kinds of information have to be associated with. And, for each type of information, explicit or tacit knowledge is linked. An efficient management of this enormous net of records depends on the skills of professionals, which are now supported by the application developed by our group.

There are other working teams that could be benefited by our approach. Besides telecommunication services managers, sales and marketing departments have to understand a client's network quite well in order to provide new solutions. The technical assistance group also needs a deep comprehension of consumer's services in order to repair possible defects. Therefore, we intend to extend the software to also support other activities within the enterprise. The telecommunication offers more than 100 kinds of services. Many of them are small variation of other services. A goal is to enlarge the application to support all of them. Finally, taking into consideration the huge amount of data and knowledge that is held by the software, managerial reports can be generated, providing useful understanding of the clients' needs in order to establish the organization market directives.

5. Conclusions

This work presented an XML-based approach to support knowledge identification, capturing and maintenance. The proposed mechanism is very efficient in distributed and heterogeneous environments since it easily integrates data, related information and associated knowledge from different sources. The proposed XML-approach represents diverse data through distinct files containing tags to identify the data and facilitate its understanding. An importation process associates the incoming data with

stored procedures placed in the main repository. As the files are described with XML, they can be parsed by any corporate system. This attribute makes the knowledge capture independent of any information structure and chosen technology.

The insertion, update, or removal of data and related knowledge are easily performed. When a new aspect of an existing understanding is identified, a corresponding tag and stored procedure are created. Neither the importation process, nor the software program that supports it has to be adapted. Therefore, knowledge management is facilitated. The proposed approach has been used for more than a year. The aggregation of novel knowledge has become an easy task to the professionals of the telecommunication company. The developed software also provides the advantage of managing the organizational memory.

6. References

- [1] C. Maidantchik, M. Montoni and G. Santos, "Learning Organizational Knowledge: An Evolutionary Proposal for Requirements Engineering", In: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, July 2002, pp. 151-157.
- [2] K. Emam., S. Quintin and N. Madhavji, "User Participation in the Requirements Engineering Process: An Empirical Study", Requirements Engineering Journal, Springer-Verlag, v. 1, 1996, pp. 4-26.
- [3] P. Hendriks and D. Vriens, "Knowledge-based systems and knowledge management: Friends or foes?" Information & Management, v. 35, n. 2 (Feb), 1999, pp. 113-125.
- [4] T. Davenport and L. Prusak, "Working Knowledge: How Organizations Manage What They Know", Boston, USA, 1998, Harvard Business School Press.
- [5] G. Fischer and J. Ostwald, "Knowledge Management: Problems, Promises, realities, and Challenges", IEEE Intelligent Systems, v. 16, n. 1 (Jan/Feb), 2001, pp. 60-72.
- [6] M. Alavi and D. Leidner, "Knowledge Management Systems: Emerging Views and Practices from the Field", In: Proceedings of the 32nd Hawaii International Conference on System Sciences, Maui, Hawaii, Jan, 1999.
- [7] G. Probst, S. Raub and K. Romhardt, "Managing Knowledge: Building Blocks for Success", 368 pp, 1999, ISBN: 0-471-99768-4.
- [8] A. Rabarijaona, R. Dieng, C. Olivier and R. Quaddari, "Building and Searching an XML-Based Corporate Memory", IEEE Intelligent Systems, v. 15, n. 3 (May), 2000, pp. 56-63.
- [9] J. Cook, "XML Sets Stage for Efficient Knowledge Management", IT professional, v. 2, n. 3 (May/Jun), 2000, pp. 55-57.
- [10] R. Dieng, "Knowledge Management and the Internet", IEEE Intelligent Systems, 2000, pp. 14-17

Sense-and-Respond Grid

J. Jeng, H. Chang, J. Chung, J. Schiefer, L. An, L. Zeng

*IBM T.J. Watson Research Center
Yorktown Heights, New York, U.S.A.*

Abstract. This paper proposes the framework for building the Sense-and-Respond (S/R) Grid in order to support a complete functionality to sense, interpret, predict, automate and respond to business activities and aims to decrease the time it takes to make the business decisions. Such infrastructure by default subsumes the hardware-based feedback loop in the common control systems. Actually, there should be almost zero-latency between the cause and effect of a business decision. The S/R grid enables analysis across corporate business processes, notifies the business of actionable recommendations or automatically triggers business operations, effectively closing the gap between business intelligence systems and business processes..

1. Introduction

Enterprises are striving to be adaptive since there are growing needs of increasing the visibility and responsiveness of business solutions due to the pressure from market and competitors. A well-accepted means of achieving an adaptive enterprise is using *Sense-and-Respond (S/R)* paradigm to monitor and manage business solutions in the enterprise [1]. In general, an S/R system can be categorized as a system that is continually interacting with its represented business organization and assisting organizational agents (software or humans) to make right decisions at right time. Such system is capable of autonomous actions in order to meet its business commitments. However, to build S/R systems is a big challenge.

The early development of Grid technologies was motivated by the problems of creating scientific resource sharing applications, e.g., collaborative visualization of large scientific data sets, and increasing functionality and availability by coupling scientific instruments and remote computer and archives [2]. Grid promises to offer solutions to the construction of reliable, scalable, and distributed systems, all of which are very important characteristics of S/R systems. The goal of this work is to combine the features from two worlds, S/R systems and Grid based computing systems, to create the S/R grid for enterprise to be extremely adaptive in the complex business environment.

The S/R operations are implemented by grid services, which export the management capabilities, intermediate execution results, state information and resource utilization information to facilitate the real-time control of Sensor-and-Respond systems. The S/R grid supports a complete functionality to sense, interpret, predict, automate and respond to business activities and aims to decrease the time it takes to make the business decisions. Such infrastructure by default subsumes the hardware-based feedback loop control in the autonomic computing framework. Actually, there should be almost zero-latency between the cause and effect of a business decision. The S/R grid enables analysis across corporate business processes, notifies the business of actionable recommendations or automatically triggers business operations, effectively closing the gap between business intelligence systems and business processes. In particular, the S/R grid addresses the issues of large scale distributed business solutions and resource management. The organization of this paper is as follows. Section 2 gives an account of the conceptual framework. Section 3 presents the high-level architecture of the S/R grid and the basic components. Section 4 shows an example of implementation. Section 5 mentions some related efforts. Section 6 presents the future work and conclusion.

2. Sense-and-Respond Framework

The S/R grid comes from two domains: Grid computing and S/R systems. A defining feature of Grids is the sharing and management of highly heterogeneous resources with the consideration of satisfying user's expectations on both functional and non-functional perspectives. Similarly, an S/R system needs to monitor and control business solutions. Business solutions come of many forms: business processes, legacy systems, business organizations, networks, enterprise data and so forth. Doubtless Grid based infrastructure is one of the best approaches to develop an enabling platform for creating S/R systems. The S/R grid needs to be policy-driven due to the dynamic nature of business environment. The S/R grid is an infrastructure used to construct policy driven S/R systems. The S/R grid drives the behaviour and functionality of target business solutions and business resources. As illustrated in Figure

1, the S/R grid takes business events from target business solutions, performs S/R operations and renders business actions back to business solutions. There are five functional stages of the S/R grid: Sense, Detect, Analyze, Decide and Effect. The aforementioned S/R stages are materialized into management services of the S/R grid: Event Processing Service (EPS), Metric Generation Service (MGS), Situation Detection Service (SDS), Analytics Processing Services (APS) and Action Rendering Services (ARS). **EPS** monitors and collects desired data from the business environment in the form of events. The functions of EPS include data extraction and cleansing, event filtering, event chaining, event transformation, event correlation, and creating qualified events. **MGS** receives qualified events from the EPS and calculates desired metrics based on metric generation rules. The generated metrics can be either published to message bus or stored into persistent storage. Business metrics generated by MGS can be the key performance indicators of suppliers and carriers. **SDS** receives metrics from the MGS and detects business situations and/or exceptions based on situation rules. Situations can be inferred by rules engine or calculated via normal procedural codes such as Java. **APS** enables the process of making the “optimal” decision for resolving business situations. This service covers the stages of “analyze” and “decide” in the S/R grid depicted in Figure 1. Such decision making process may be involved very complicated business intelligence modules. **ARS** renders appropriate management directives based on the decision that has been made. Examples of actions for an inventory S/R grid include modification of the data entries for target inventory policies; and sending altering messages to LOB managers.

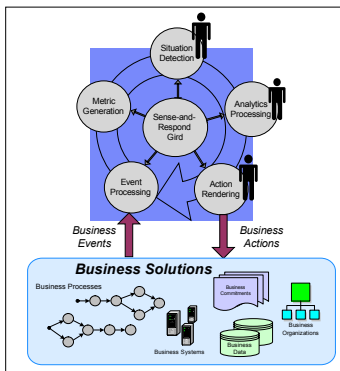


Figure 1: S/R Grid and Business Solutions.

3. Grid Based Architecture

From the point of view of adaptive enterprise, an important issue is how to enable transparent and dynamic monitoring and control of target business solutions across business organizations. In this context, the S/R grid serves as the platform spanning multiple networks, business

systems, business processes and organizations. The S/R grid facilitates secure and efficient sharing of the data and services needed for monitoring and managing business solutions. The S/R grid comprises management services, business resource managers, and business resources including business organizations, processes, documents, all other data, and accompanying applications. All of the above entities are virtualized into uniform view of grid services. The motive is to make the S/R grid more flexible, adaptive, and powerful by querying service registries at run time in order to discover information and network-addressable third-party S/R components.

For example, a computation-intensive business forecasting application looks for remote services that manifest appropriate composition of features such as matched algorithms, suitable interfaces and operations, and quality of services. Grid services can be described, discovered, matched, and used via the extension of existing Web Service based standards [3]. Contrary to conventional system management domain, the demand of monitoring and controlling business solutions needs to be sensed or even inferred from given data through quantitative metrics using monitoring centric services, i.e., EPS, MGS and SDS. The management capabilities of the S/R grid will be accomplished through response-centric services such as APS and ARS. The policies of selecting and invoking management services are described by an XML-based policy specification coined as Business Performance Commitment Language (BPCL) [4]. A document in that language describes the target business solutions with all available management services of its governing S/R grid and their relationships.

A fully-fledged S/R grid consists of four building blocks. First, it contains grid based infrastructure for business solutions. The motive of such infrastructure is simple: since business solutions are the target of the S/R grid, there should be an integrated approach of defining the monitoring and control interfaces. The most important requirement of such infrastructure is using grid services to model and wrap target business solutions. Similarly, target business resources within an enterprise require uniform treatment as grid services too. They constitute the second building block of the S/R grid, a virtualization layer of situated business resources in the enterprise. Business resources include anything supporting the functionality and behavior of business solutions: business processes, business systems, business organization, data repositories and so forth. Grid services provide the means of achieving this goal of virtualization of the above two building blocks. The third building block is the collection of all management capabilities in an integrated infrastructure. All management capabilities are described and developed as grid services. The detail account of management services will be given in next section. The

fourth building block is the controller and choreographer of the S/R grid that provisions management services, mediates their invocations, and maintains the relationships among them.

Figure 2 introduces the general architecture of the S/R system which consists of 5 layers: (1) A layer of virtual business resources (VBR), made up of the business resources provided by the enterprise and its partners, provisions resource specific monitoring and control interfaces. (2) A layer of virtual business solutions (VBS), made up of the business solution grid services, provisions solution specific monitoring and control interfaces. Business solutions signify the source of management demands that can be either pulled or pushed to an S/R system. (3) A layer of virtual management capabilities (VMC), made up of the management services, provisions management capabilities and performs capability-to-service mapping. (4) A layer of virtual business organizations (VBO), made up of S/R services, each of which represents a business organization. Here, commitment is referred to as business commitment. A business organization grid service makes decisions and enforces them to resolve business situations through its governed business resources and management services.

There are many-to-many relationships between business solutions and business organizations. AN S/R controller possesses an overlay-structure S/R network of nodes that maintain information about business organizations, business solutions and their relationships. This network structure reflects the business relationships in the context of S/R domain. Such relationships can be either logical or physical, and are defined by business commitments.

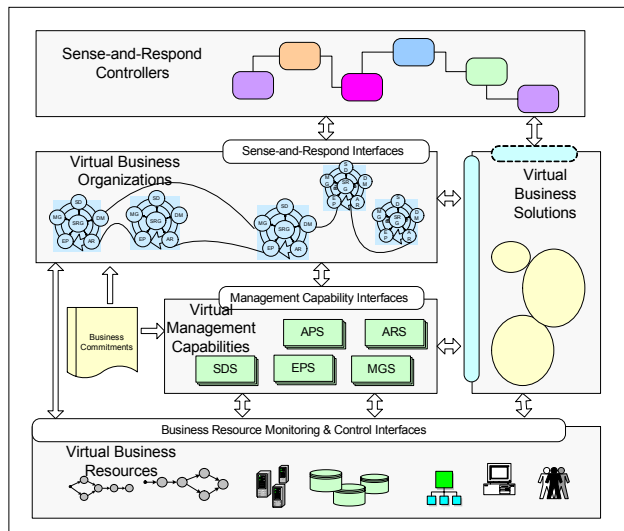


Figure 2: The Architectural for S/R Grid.

4. Implementation and Validation

We are working with a chip maker and applied the S/R to fulfill the requirements raised in the domain of electronic manufacturing. Specifically, we are implementing an S/R system based on the framework described in this paper. This system senses events generated from the manufacturing organization, detects manufacturing-related business situations, conducts analysis on the data embedded in the situations and enterprise database, provides recommended actions to decision makers and finally renders business actions to the target business solution and systems. Automated software agents materialize business actions into system-level actions and realize them to the target business systems. Between the dashboard and the S/R grid, there is a presentation layer with different grid services. Figure 3 presents the unified view of a business dashboard for the S/R system users to monitor the manufacturing processes and activities, manufacturing exceptions, links to perform OLAP analysis, presents recommended actions to manufacturing exceptions and so on. The right-hand side of this console presents four monitoring portals and the real-time monitoring of events are shown in the portal on the left-hand side.

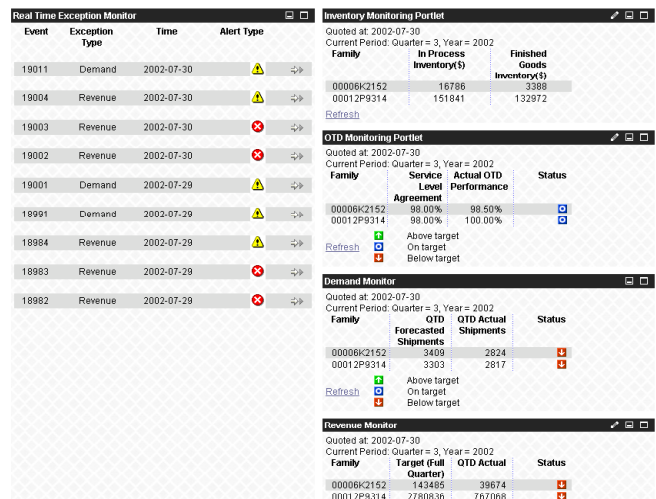


Figure 3: S/R Dashboard for Processes.

Figure 4 shows the revenue performance of a manufacturing process. The portal on the left-hand side shows the revenue statistics including both actual and predictive performance data. On the right-hand side, a graphical representation of the performance data is shown in a portal where the upper and lower bounds indicate the performance targets. A business situation will be raised whenever the revenue performance data is out of the boundaries. Thereafter, a decision making process will be triggered to resolve such situation. As mentioned, all of these interactions are handled by designated web services

based upon predefined business commitments and enforced by the S/R grid in the back end.

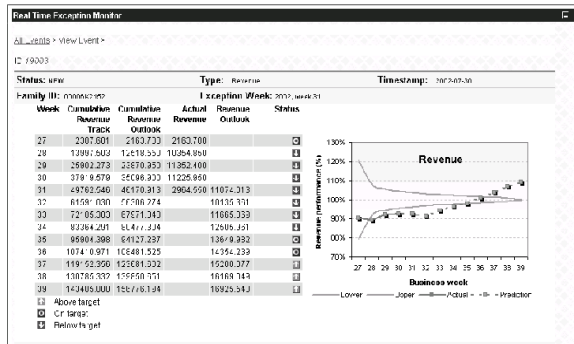


Figure 4: Monitoring Revenue Performance.

Based on our experience of using the S/R grid to implement S/R systems, we have obtained the following observations: (1) The formal model of the S/R nets makes explicit the S/R concepts and helps developers to model and develop such systems more effectively. (2) As we migrated from one the S/R project to another, we experienced shorter development lifecycle. According to our experience, the third project is witnessed to have more than 50% reduction of development cycle. (3) The S/R grid makes possible the intimate interplay between policies and runtime artefacts. Thus, the behaviour of the S/R systems can be consistently and soundly changed without much trouble of the mismatch between requirements and system behaviour. (4) The reference architecture and implementation based on the S/R grid have proven to be a group of reusable assets for creating Grid-based S/R systems. (5) The S/R grid actually renders the patterns of monitoring and controlling service-oriented business solutions, which can be very valuable for future direction of Grid implementation. In reality, we are trying to standardize the S/R specification into standards bodies.

5. Related Work

IBM's Autonomic Computing [5] vision aims to provide self-managing systems to create systems that respond to capacity demands and system failures without human intervention. The S/R grid is aimed to achieve the similar purpose at the business level and extend it into global-scale service grids. Traditional grid based approach [6] such as Globus (www.globus.org) focus on distributed supercomputing, in which schedulers make decisions about where to perform computational tasks. The S/R grid is governed by business commitments that are a mode sophisticated policies than OGSA policy. Graupner et al. [7] proposed an architecture for automated demand-supply control system based on a formalization of service

demands and supplies in an overlay meta-system. Although the approach of using S/R to mediate the management capabilities (supply) and business situations (demand) is similar to their approach on meta-systems, we are taking a policy driven approach to creating S/R grids.

6. Future work and Concluding Remarks

In this paper, we advocate using grid service to implement the S/R systems. We are investigating how the S/R system in the distributed business solutions utilizes the S/R grids for allocating computing resources and addressing load balance (adaptive resource distribution) to routing business events among service components and to conduct business data-mining and mathematical analysis. We are developing the presented approach as a research project under the mission of establishing globally distributed S/R grids for adaptive enterprise. The idea is to automate management tasks with the ultimate goal of achieving extremely adaptive behavior, a vision IBM calls on demand business. The S/R grid is one step toward that goal.

References

- [1] S.H. Haeckel, and A.J. Slywotzky, *Adaptive Enterprise: Creating and Leading S/R Organizations*, Harvard Business School Publisher, August, 1999.
- [2] I. Foster, and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco, 1999.
- [3] E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, Web Services Description Language (WSDL) 1.1 *W3C Note 15*, 2001. <http://www.w3.org/TR/wsdl>.
- [4] H. Li, J.J. Jeng, "Managing Business Relationship in E-Services Using Business Commitments", Proceedings of Third International Workshop, TES 2002, Hong Kong, China, August 23-24, 2002, LNCS 2444, pages 107-117.
- [5] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," white paper, IBM Research, 2001, <http://www.research.ibm.com/autonomic/manifesto>
- [6] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems," *Software—Practice and Experience*, vol. 32, no. 2, 2002, pp. 135–164.
- [7] S. Graupner, V. Kotov, A. Andrzejak, and H. Trinks, "Service Centric Globally Distributed Computing," *IEEE Internet Computing*, Vol. 7, No. 4, pp. 36-43, July-August, 2003.

The KAMET II Architecture for Problem-Solving Method Reuse

Instituto Tecnológico Autónomo de México (ITAM)

Osvaldo Cairó

cairo@itam.mx

Julio César Álvarez

calvarez@alumnos.itam.mx

Abstract. Problem-solving methods are ready-made software components that can be assembled with domain knowledge bases to create application systems. In this paper, we describe this relationship and how it can be used in a principled manner to construct knowledge systems. We have developed a methodology that strongly relies on ontologies: first, to describe domain knowledge bases and problem-solving methods as independent components that can be reused in different application systems; and second, to mediate knowledge between the two kinds of components when they are assembled in a specific system. We present our methodology and a set of associated tools that have been created to support developers in building knowledge systems and that have been used to conduct problem-solving method reuse.

1. Ontologies

Ontologies provide a structured framework for modeling the concepts and relationships of some domain of expertise. Ontologies support the creation of repositories of domain-specific reference knowledge -*domain knowledge bases*- for communication and sharing of this knowledge among people and computer applications. Ontologies provide the structural basis for computer-based processing of domain knowledge to perform reasoning tasks. In other words, ontologies enable the actual use of domain knowledge in computer applications. Problem-Solving Methods provide reusable reasoning components that participate in the principled construction of knowledge-based applications.

2. Problem-Solving Methods

Overcoming the limitations of both rule-based systems and custom programs, Problem-Solving Methods (PSMs) were introduced as a knowledge engineering paradigm to encode domain-independent, systematic and reusable sequences of inference steps involved in

the process of solving certain kinds of application tasks with domain knowledge. KAMET Architecture

The KAMET II Methodology [1, 2] relies on a conceptual and formal framework for the specification of knowledge-based systems. This conceptual framework is developed in accordance to the CommonKADS model of expertise [5]. The formal means applied are based on combining variants of algebraic specification techniques and dynamic logic [4]. The framework consists of the following elements: a *task* that defines the reasoning process of a knowledge-based system, a *problem-solving method* that defines the reasoning process of a knowledge-based system, and a *domain model* that describes the domain knowledge of the knowledge-based system. Each of these elements is described independently to enable the reuse of tasks descriptions in different domains. Therefore, a fourth element of a specification of a knowledge-based system is an *adapter* which is necessary to adjust the three other (reusable) parts to each other and to the specific application problem. Fig. 1 shows the architecture. [4] gives a detailed explanation of these elements.

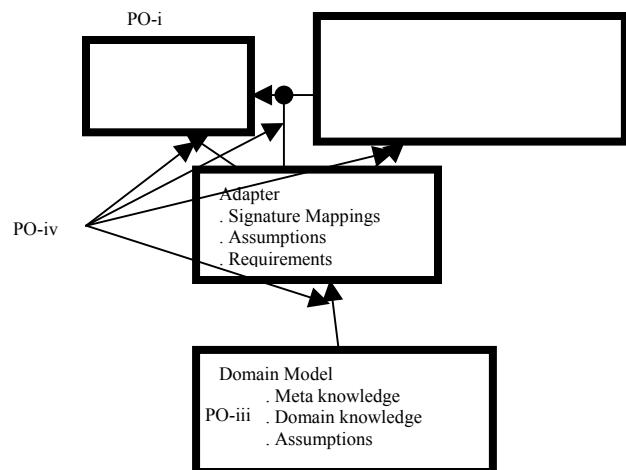


Fig. 1 The KAMET II Architecture

3. An Ontology-based Approach to Developing Knowledge Systems

The use of ontologies in constructing a knowledge system is pervasive. At least, ontologies support the modeling of the domain-knowledge component counterpart of PSMs in knowledge applications. However, PSMs and domain ontologies are developed independently and therefore need to be reconciled to form a coherent knowledge system. As the basis for reconciliation, PSMs declare the format and semantics of the knowledge that they expect from the domain to perform their task [3]. A PSM provides a method ontology, that elicits its input and output knowledge requirements, independently of any domain. For instance, the generate-and-test PSM declares its input-knowledge needs in terms of state variables, constraints and fixes. This way, the method ontology assigns roles that the domain knowledge needs to fill so that the PSM can operate on that knowledge. Further, the method ontology states the assumptions that the PSM makes on domain knowledge. Besides making all domain knowledge requirements explicit, refined versions of the PSM can be modeled directly by weakening or strengthening its assumptions by way of additional sets of ontological statements - or adapter component [3].

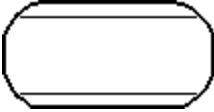
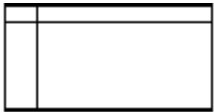
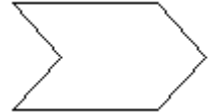
	Problem. Expresses an alteration, disorder or abnormality.
	Classification. Expresses alterations, disorders or abnormalities that can be considered a <i>classification</i> problem. Therefore, it can be represented within a table.
	Subdivision. Expresses an alteration, disorder or abnormality that can be subdivided into smaller problems.

Table 1. Structural Constructors

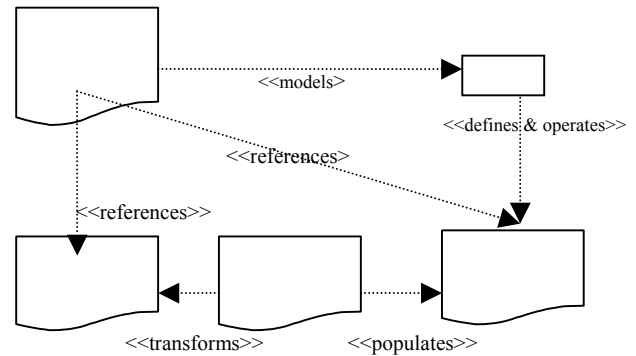


Fig. 2 Use of Ontologies in KAMET

To avoid impairing the independence of either the domain or the method ontologies, this approach includes a mediating component. This third, separate knowledge component holds the explicit relationships between the domain and the method ontologies assembled in a specific knowledge application [3]. Underlying this mediating component is a mapping ontology that bridges the conceptual and syntactic gaps between the domain and method ontologies. [3] studies this in depth.

4. Modeling a Diagnosis Task in KAMET

This section presents an example of how to model a diagnosis task by means of KAMET II. We will take a simple economic problem: how to determine if a country's economy is going through a recession. The definition of recession is the prolonged period of time when a nation's economy is slowing down or contracting. This period might go from six months to two years. The economy is the production and consumption of goods and services. Before continuing, the symbols of KAMET are presented in order to make the example comprehensible. The symbols of KAMET are used for modeling visually knowledge models and problem-solving methods and they are the means by which the concepts needed by ontologies are modeled. The KAMET II CML has three levels of abstraction. The first corresponds to structural constructors and components. The second level of abstraction corresponds to nodes and composition rules. The third level of abstraction corresponds to the global model [2]. Table 1, 2 and 3 present them.

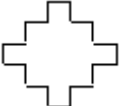
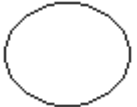



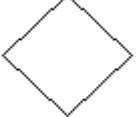
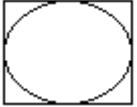
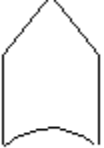
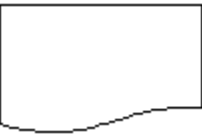


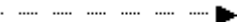
	Symptom. Manifestation or signal related to an alteration, disorder or abnormality.
	Antecedent. Expresses previous circumstances that can be used to judge something that can happen.
	Solution. Expresses possible solutions to a disorder, alteration or abnormality. It is always related to structural constructors.
	Time. Expresses the duration of structural components (symptoms and antecedents) as web of structural constructors (problems and subdivisions).
	Value. Expresses characteristics of symptoms, antecedents or groups.
	Inaccuracy. Expresses uncertainty due to the lack of precision of an intermediate or terminal node.
	Process. Expresses the sequence of actions and operations required to obtain a result.
	Formula. Expresses calculus that must be completed in order to determine the alteration, disorder or abnormality.
	Examination. Expresses a recommendation or necessity of making studies, examinations, proofs for determining an alteration, disorder or abnormality.

Table 2. Structural Components

	Division. Expresses that an alteration, disorder or abnormality is subdivided into...
	Implication. Expresses connection between a cause and a complication.
	Action. Expresses that

something must be completed, a formula or an exam.

Union. Expresses connections between subdivisions.

Table 3. Composition Rules.

Fig. 3 shows the causal chain that explains the behaviour of a nation's economy.

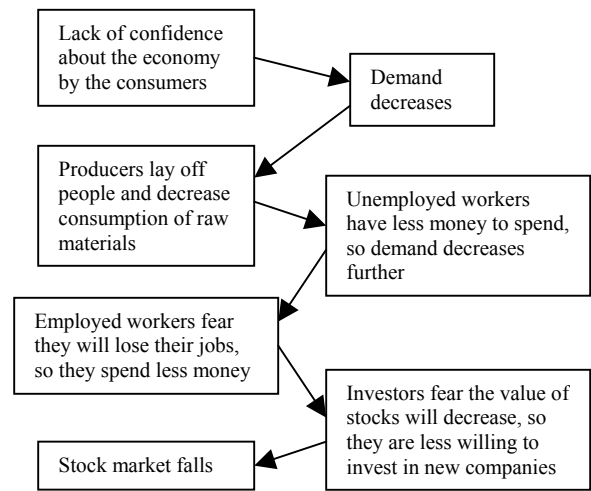


Fig. 3. When things go wrong

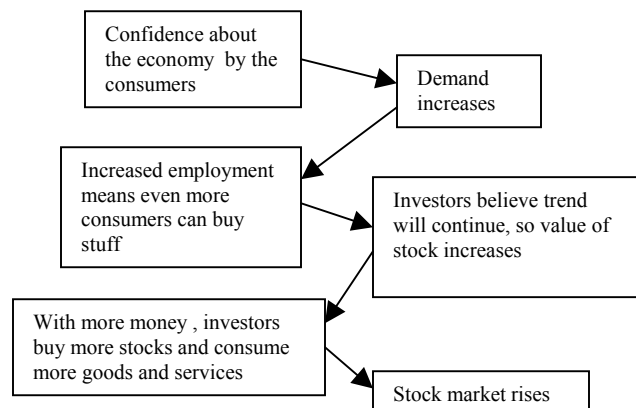


Fig. 4 When things go well

Below it is presented the knowledge base of this application using the KAMET II Conceptual Language.

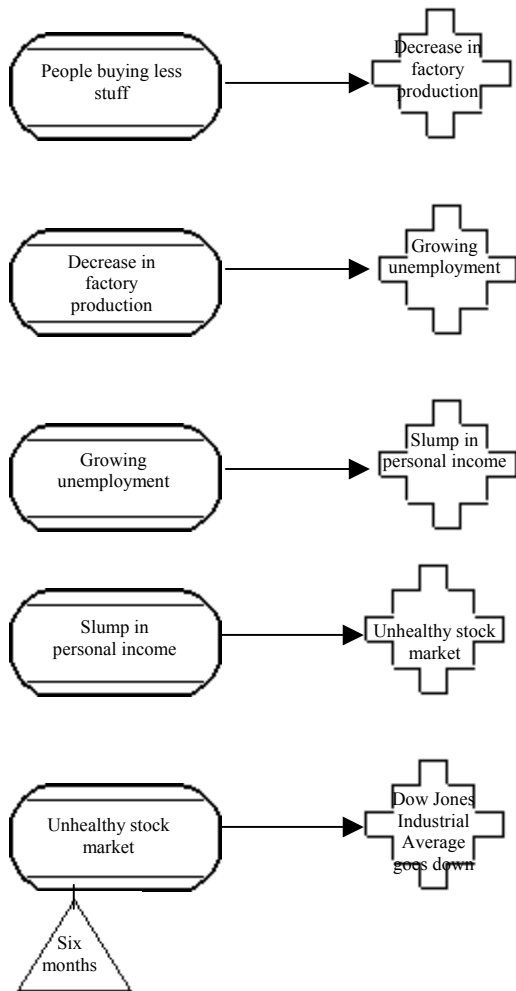


Fig 5. Brief economy knowledge base

We present an ontology description in Fig. 5.

Domain model *economy*

Signature

Sorts *hypothesis, hypotheses* **set of** *hypotheses,*
symptom

Predicates

Causes: hypothesis x symptom

Variables

h: hypothesis

s: symptom

H, H' : hypotheses

Domain knowledge

Causes (People buying less money, decrease in factory production)

Causes (decrease in factory production, growing unemployment)

Causes (growing unemployment, slump in personal income)

Causes (slump in personal income, unhealthy stock market)

Causes (unhealthy stock market and six months, index decreasing)

Enddm

Fig 6 The domain model

The diagrammatic representation in KAMET of *generate-and-test* is presented below.

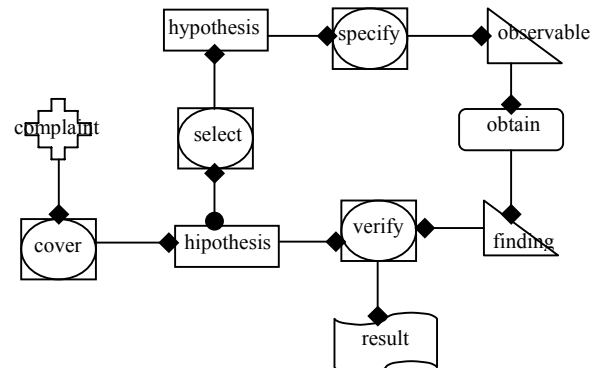


Fig 7 The problem-solving method

5. Conclusions

We presented the KAMET II capabilities for modeling domain knowledge as well as for modeling reasoning knowledge. We showed the architecture that enables domain knowledge and problem-solving knowledge reuse.

References

1. Cairó, O.: A Comprehensive Methodology for Knowledge Acquisition from Multiple Knowledge Sources. *Expert Systems with Applications*, 14(1998).
2. Cairó, O.: The KAMET Methodology: Content, Usage and Knowledge Modeling. In Gaines, B. and Musen, M., editors, *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 1-20. Department of Computer Science, University of Calgary, SRGD Publications
3. Crubézy, M., and Musen, M.: Ontologies in Support of Problem Solving. *Handbook on Ontologies in Information Systems*. S. Staab and R. Studer, Springer.
4. Fensel, D.: Problem-Solving Methods Understanding, Description, Development, and Reuse. *Lecture Notes in Artificial Intelligence*, Vol. 1791. Springer-Verlag.
5. Schreiber, G.: *Knowledge Engineering and Management: the CommonKADS Methodology*. MIT Press, Cambridge, Massachusetts, 1999.
6. Schreiber, G., Crubézy, M., and Musen, M.: A Case Study in Using Protégé-2000 as a Tool for CommonKADS. In Dieng, R. and Corby, O., editors, *12th International Conference, EKAW 2000*, Juan-les-Pins, France.

Using COSMIC-FFP for Predicting Web Application Development Effort

G. COSTAGLIOLA, F. FERRUCCI, C. GRAVINO, G. TORTORA, G. VITIELLO

Dipartimento di Matematica e Informatica, Università degli Studi di Salerno

Via Ponte Don Melillo, 84084 Fisciano (SA), Italy

email: {gcostagliola, fferrucci, gravino, tortora, gvitiello}@unisa.it

Abstract. *In the paper we provide a set of rules to apply the COSMIC-FFP method for measuring the size of dynamic web applications. The rules can be applied to analysis and design documentation in order to provide an early estimation. We also describe the empirical analysis carried out to verify the usefulness of the method for predicting web application development effort. Such analysis provides promising results encouraging us to further investigate the validity of the approach.*

1. Introduction

In the last years the demand for web applications is quickly increasing, since they are an essential support for the activities of organizations which operate in various areas. The complexity and size of such applications have also dramatically augmented. Thus, there is the need for tools supporting project development planning with reliable cost and effort estimations.

In the context of traditional software engineering many software measures have been defined to gather information about relevant aspects of software products and then manage their development. In particular, several size measures have been conceived to be employed in effort/cost models to predict the effort and cost needed to design and implement the software (see, e.g., [1, 6]). When dealing with web applications such measures turn out to be rather inadequate failing to capture some specific features which significantly affect the size and then the effort required for those applications [8,9,11,12]. Nevertheless, many researchers agree that some existing methods can be generalized and or adapted in order to be successfully used for measuring size of web information systems. In particular, Rollo considered *COSMIC-FFP* (*cosmic full function point*) [5], which represents an adaptation of the *Function Point* method [1], especially focused on data movements. Although specifically devised to tackle real-time and embedded applications, *COSMIC-FFP* turned out to be able to capture the functional size of other systems such as Management Information Systems. In [9] Rollo

applies the measure to an Internet Bank System and suggests its use in the context of web based applications. Following his suggestion in [7] Mendes et al. provide a formalization of the method for hypermedia web systems and report on an initial statistical analysis which has been carried out on systems designed and authored by students.

In this paper we propose an adaptation of *COSMIC-FFP* taking into account dynamic web applications. Indeed, the measure turns out to be suitable for capturing also the dynamic aspects of such applications which are characterized by data movements to and from web servers. A formalization of the adaptation is provided by suitably revising some basic concepts of the method and defining appropriate procedures to measure the functional size of software by counting the data movements. Such procedures have been conceived to be applied on design documents, such as use cases and class diagrams in order to provide an early size estimation. An encouraging, yet initial, empirical validation of the measure has been gained by applying an Ordinary Least-Squares (OLS) regression analysis on a set of dynamic applications developed by undergraduate students of an academic course on web engineering.

The paper is organized as follows. In Section 2 we recall the main concepts of the *COSMIC-FFP* method, and explain how we have adapted it for web applications. An example of use of the method is also illustrated. Section 3 presents the results of the empirical analysis carried out so far. Section 4 concludes the paper giving some final remarks and discussion on future work.

2. The method

In the present section we recall the main concepts of the *COSMIC-FFP* method, and explain how we have adapted it for web applications, by providing a set of rules specifically conceived to measure the size of dynamic web applications. Example of use of those rules are also described.

COSMIC-FFP involves to apply a set of models, rules and procedures to *Functional User Requirements* to obtain a numerical value which represents the functional size of the software, expressed in terms of *CFSU* (*cosmic functional size unit*) [5]. In order to apply the method, two models are

identified: the *context model* and the *software model*. The former establishes what is part of the software to be sized and what is part of the software's operating environment (see Fig. 1), by identifying boundaries and illustrating the generic functional flow of *data attributes* from a functional perspective. The flow of *data attributes* is characterized by two directions, *back-end* and *front-end*, and by four distinct types of movements: *entries* and *exits*, which allow the exchange of data with user, and *reads* and *writes*, which allow the exchange of data with the persistent storage hardware.

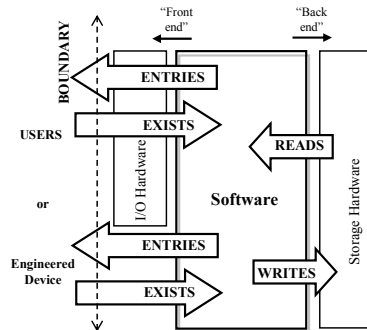


Figure 1. Generic flow of data attributes from functional perspective [5]

The *software model* assumes that two general principles hold for the software to be mapped and measured: 1) software takes input and produces useful output to users, and 2) software manipulates pieces of information designated as data groups which consist of data attributes.

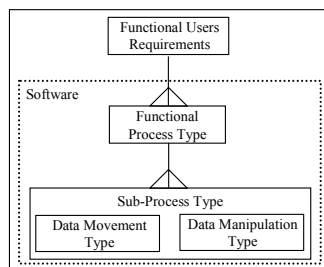


Figure 2. A generic software model for measuring functional size [5]

Such *software model* allows us to consider the functional user requirements decomposed in a set of functional processes, where each process is a unique set of sub-processes performing either a data movement or a data manipulation (see Fig. 2). The data movement sub-processes *entry*, *exit*, *read*, *write*, which move data contained in exactly one data group, are considered. The functional size of software is directly proportional to the number of its data movement sub-processes. Such an assumption is justified by the nature of the software the

method was initially targeted at, namely real time applications, which are characterized by several movements of data. The consideration on the fact that dynamic web applications are also characterized by data movements (from a web server to the client browser), has suggested us to apply the principles of the *COSMIC-FFP* method to size this type of web applications in terms of their functionality. To this aim, the *context model* and the *software model* have been suitably adapted. Moreover, appropriate procedures to count the data movement sub-processes have been defined that can be applied on design documents, such as use cases and class diagrams. When dealing with web applications, the flow of data attributes gives rise to the *context model* illustrated in Fig. 3. Web applications are bounded in the *back-end* direction by the web server which allows us to interact with the information stored in the server and to communicate with the web browser to provide the requested documents. In the *front-end* direction web applications are bounded by I/O hardware and users are humans which interact with web applications across the boundary¹.

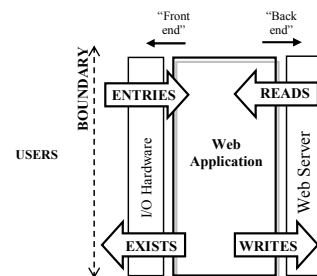


Figure 3. The functional flow of data attributes through web application

Fig. 4 depicts the generic *software model*, where the data movement sub-processes are identified by analyzing use cases and class diagrams obtained from functional user requirements.

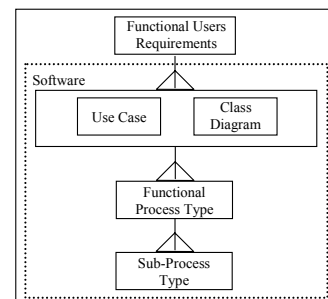


Figure 4. A generic software model to measure functional size of web applications

¹ It is worth noting that a similar model was provided by Mendes et al. in [6], where the *write* data movement type was missing since only static web applications were considered.

In order to provide the rules to measure data movements let us identify the possible components of a web application as follows.

- **Multimedia Component:** able to generate output in multimedia format (e.g., graphics, audio, video).
- **Server-Side Dynamic Component:** able to provide dynamic functionality on the server side. The two main approaches are compiled modules and interpreted scripts.
- **Client-Side Dynamic Component:** scripts and applications used to build web applications and automate their functionality on the client side.
- **External Reference:** used to provide a reference to external applications.

In order to count the data movement sub-processes (*entry*, *exit*, *read* and *write*) we have provided the following rules.

1. For each static web page count 1 *entry* + 1 *read* + 1 *exit*. Indeed, an entry is sent to the application by requesting the client page (*entry*), the page is read from the web server (*read*) and then shown to the user (*exit*).
2. For each multimedia component, which is visualized after an explicit request of the client, count $C \cdot (1 \text{ entry} + 1 \text{ read} + 1 \text{ exit})$. In other words, the media is considered as another web page downloaded from the server when it is requested. C denotes a weight associated to the component and is determined by considering its influence on the development process. In particular, $C=1$, means little influence; $C=2$, means medium influence; $C=3$, means strong influence.
3. For each script used to provide a functionality to manipulate document on the client side (i.e., in the web browser), count 1 *entry*.
4. For each application executed on the client side, count $C \cdot (1 \text{ entry} + 1 \text{ exit})$. The *entry* is considered to run it and the *exit* to show it. Again C denotes a weight associated to the component and is determined by considering its influence on the development process. In particular, $C=1$, means little influence; $C=2$, means medium influence; $C=3$, means strong influence.
5. For each server side interpreted script or compiled module used to produce a dynamic web page, count 1 *entry* + 1 *read* + 1 *exit*. In this case, a form allows users to input data and request a dynamic page (*entry*). The web server elaborates the input of the user through the server-side script or module (*read*) and produces a web page which is sent to the user (*exit*). Moreover, count an additional *read* if an access control is first performed (e.g., for checking login and password) and then the input is elaborated to generate the web page.
6. For each server side script modifying persistent data through the web server, count 1 *entry* + 1 *write* + 1 *exit*. The user inputs data through a form (*entry*), the data is

written through the web server (*write*) and the result is shown to the user (*exit*). Count an additional *read* if an access control is first performed.

7. For each web page that contains confirmation, alert or error messages sent by the web server to the browser, count 1 *read* + 1 *exit*.
8. For each reference to external applications such as commercial package, library routine, count 1 *entry* + 1 *exit*.

Let us note that rules 5, 6, 7, 8 are specifically conceived to consider dynamic aspects of web applications, rule 2 refers to multimedia components and rules 1, 3, 4 take into account elements common to static web applications. In particular, the latter rules are analogous to the ones provided by Mendes et al. in [7] to measure hypermedia web applications.

To determine the functional size of a web application, the corresponding use case and design documents are analyzed in order to apply the above rules. The resulting sum is expressed in terms of *CFSU*.

An example of application of the proposed method

In the sequel we show the application of the rules for counting data movements. To this aim, let us consider the class diagram depicted in Fig. 5, which is referred to a web application designed for e-learning purposes.

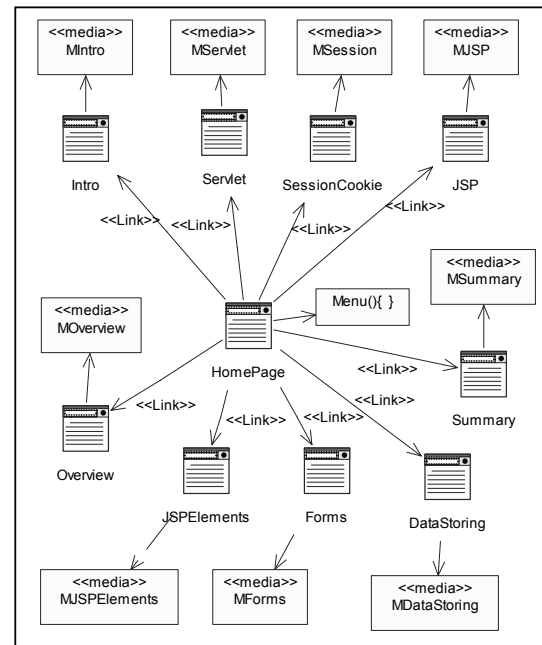


Figure 5. The UML class diagram modelling the activities for a e-learning course

The diagram adopts the UML notation for the web proposed in [3] which exploits *stereotypes*, *tagged values* and *constraints* to suitably denote components that are particular to web applications such as *server pages*, *client page*, *page*, *form*, *frameset*, *client script*, etc. In Fig. 6 the icons denoting some of these components are depicted.

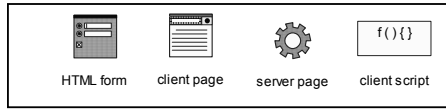


Figure 6. Icons representing web components according to Conallen's UML extension

The class diagram in Fig. 5 models the activities for presenting learning objects in a distance course. From the client page *HomePage*, 9 client pages can be accessed, namely *Intro*, *Servlet*, *SessionCookie*, *JSP*, *JSPElements*, *Forms*, *DataStoring*, *Overview* and *Summary*. Each of those web pages contains a request for a media which is specified by the stereotype `<<media>>`. Moreover, *HomePage* contains a client script.

By applying rule 1 we obtain 30 *CFSUs* due to the presence of 10 client pages. The presence of a client script in the *HomePage* determines the application of rule 3, and then one more *CFSU*. Finally, the application of rule 2 determines further 81 *CFSUs*, since 9 media are requested by the client pages, with an estimated weight $C=3$. Thus, for this class diagram we have a total of 112 *CFSUs*.

Now, let us analyze the class diagram modelling the final test activities for the given learning object (see Fig. 7). The description of the corresponding use case (see Fig. 8) can further support us in the comprehension of the diagram and in the identification of data movements.

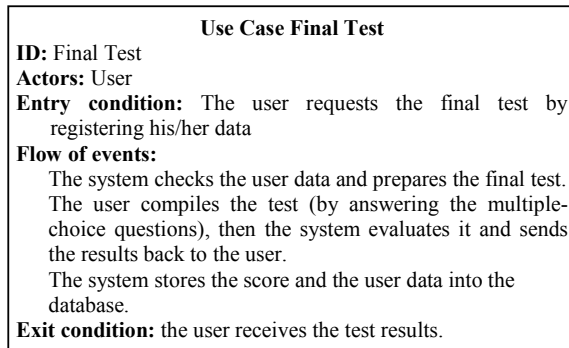


Figure 7. The use case Final test

The user requests the final test by specifying his/her data through the HTML form *UserRegistration* contained in the client page *FinalTest*. The server page *UserIdentification* verifies whether or not the user is registered and the server page *TestCreation* prepares the form *TestForm* by using the information of the class *Test*. The user fills in the form by

answering the questions and submits his/her test. Then, the server page *Scoring* interacts with the database and determines the score which is sent back to the user as an HTML page (i.e., *Score*). Moreover, the server page *DBUpdating* inserts the score into the database by using the user data contained in the object *Session*.

The presence of the server pages *UserIdentification*, *TestCreation*, *Scoring* determines the application of rule 5, resulting in 9 *CFSUs*. Rule 6 is instead applied considering the server page *DBUpdating*, determining other 3 *CFSUs*. Finally, the presence of the static web page *FinalTest* which contains the HTML form *UserRegistration*, causes the application of rule 1, counting further 3 *CFSUs*. Thus, the total counting for the considered piece of design documentation is 15 *CFSUs*.

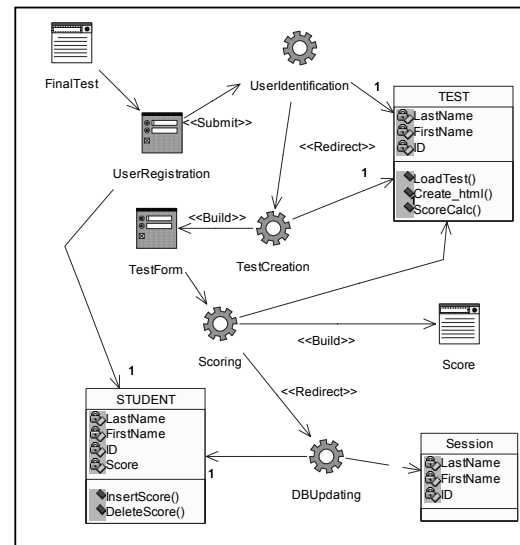


Figure 8. The UML class diagram modelling the final test

3. Empirical Evaluation

A statistical analysis has been performed to establish whether the proposed adaptation of *COSMIC-FFP* can be used to predict web application development effort in terms of person-hours. We have exploited data coming from 22 web projects developed by students during an academic course on web engineering. Students were instructed on web design by using Conallen's UML extension and on common web technologies. In order to allow uniformity, the most skillful students were equally distributed among the 22 groups. Each group was asked to design and implement a client-server hypermedia application and to record information on the actual effort required for the development process in terms of person-hours.

Table 1 reports the data of the 22 projects. A descriptive statistics has been performed both for the variable *Effort*

(denoted by *EFH*), expressed in terms of person-hours, and the variable *COSMIC-FFP* (denoted by *C-FFP*), expressed in terms of *CFSUs*, related to the 22 systems used. The summary statistics of those variables are given in Table 2.

Table 1. The data for the 22 web development projects

OBS	EFH	C-FFP	OBS	EFH	C-FFP
1	110	165	12	128	430
2	108	250	13	145	600
3	141	519	14	133	483
4	104	417	15	148	543
5	118	311	16	119	255
6	120	298	17	125	380
7	152	612	18	153	577
8	135	475	19	159	263
9	131	355	20	171	807
10	105	187	21	163	778
11	135	401	22	172	833

Table 2. Descriptive statistics of *EFFORT* and size expressed in *C-FFP*

	OBS	MIN	MAX	MEAN	STD. DEV.
EFH	22	104	172	135,2273	22,8119
C-FFP	22	165	833	451,7727	193,3758

In order to perform the empirical validation of the proposed method, we have applied an Ordinary Least-Squares regression analysis.²

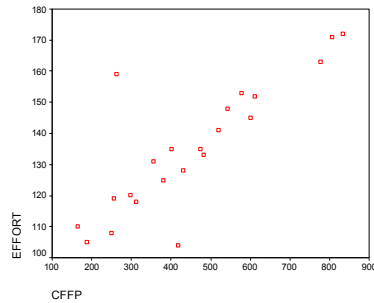


Figure 9. The scatter plot for *EFH* and *C-FFP*

Fig. 9 illustrates the scatter plot obtained by considering *EFH* as dependent variable and *C-FFP*, as independent variable. The scatter plot shows a positive linear relationship between the variables involved. This suggests that a linear regression analysis of *EFH* and *C-FFP* can be performed. The linear regression analysis allows us to determine the equation of a line, which interpolates data and can be used to predict the development effort in terms of the number of person-hours required. The goodness of fit of a regression model is determined by the square of the linear correlation coefficient, R^2 . We can

observe that the linear regression analysis shows a high $R^2=0,690$, which indicates that 69% is the amount of the variance of the dependent variable *EFH* that is explained by the model related to *C-FFP*. Moreover, the F value and the corresponding p -value (denoted by *Sign F*) are useful indicators of the degree of confidence of the prediction. Specifically, in Fig. 10.a we can observe a high F value (44,617) and a low p -value (0,000), which indicate that the prediction is indeed possible with a high degree of confidence. We have also considered the p -values and t -values for the corresponding coefficient and the intercept (see Fig. 10.b). The p -values give an insight into the accuracy of the coefficient and the intercept, whereas their t -values allow us to evaluate their importance for the generated model. In particular, since for both variables the p -value is less than 0.05, the variables are significant predictors with a confidence of 5%. As for the t -value, a variable is significant if the corresponding t -value is greater than 1.5, which is the case for both the coefficient and the intercept.

R^2	R	Std Err	F	Sign F
0,690	0,831	11,8644	44,617	0,000

(a)

	Value	Std. Err	t-value	p-value
Coefficient	8,943E-02	0,013	6,680	0,000
Intercept	94,825	6,556	14,463	0,000

(b)

Figure 10. The results of the OLS regression analysis for evaluating the *EFH* using *C-FFP*

Thus, the equation of the regression model obtained with this data set is:

$$EFH = 8,943E-02 * C-FFP + 94,825$$

where the coefficient 8,943E-02 and the intercept 94,825 are significant at level 0.000, as from the T test.

In order to assess the acceptability of the derived effort prediction model, we have considered the *Magnitude of Relative Error*, which is defined as

$$MRE = |EFH_{real} - EFH_{pred}| / EFH_{real}$$

where EFH_{real} and EFH_{pred} are the actual and the predicted efforts, respectively. The rationale behind this measure is that the gravity of the absolute error is proportional to the size of the observations. Such value has been calculated for each of the 22 observations in the data set. We have evaluated the prediction accuracy by taking into account a summary measure, given by the *Mean of MRE (MMRE)*, to measure the aggregation of *MRE* over the 22 observations. The values of such measures are reported in Table 3. In particular, we can observe that the model exhibits an *MMRE* value less than 0.25. As suggested by Conte *et al* in [4], this represents an acceptable threshold for an effort prediction model.

Moreover, we have considered another meaningful measure, namely the *prediction at level l*, defined as

$$PRED(l) = k / N$$

² For our study, we have employed the package *SPSS* for Windows, release 9.0.

where k is the number of observations whose MRE is less than or equal to l , and N is the total number of observations. Again, according to Conte *et al.*, at least 75% of the predicted values should fall within 25% of their actual values. In other words, a good effort prediction model should have $PRED(0.25) \geq 0.75$. Also this condition turns out to be satisfied by the derived model.

Table 3. The validation results

OBS	EFF _{real}	EFH = 8,943E-02 * C-FFP + 94,825		
		C-FFP	EFF _{pred}	MRE
1	110	165	109,581	0,0038
2	108	250	117,1825	0,0850
3	141	519	141,2392	0,0017
4	104	417	132,1173	0,2704
5	118	311	122,6377	0,0393
6	120	298	121,4751	0,0123
7	152	612	149,5562	0,0161
8	135	475	137,3043	0,0171
9	131	355	126,5727	0,0338
10	105	187	111,5484	0,0624
11	135	401	130,6864	0,0320
12	128	430	133,2799	0,0412
13	145	600	148,483	0,0240
14	133	483	138,0197	0,0377
15	148	543	143,3855	0,0312
16	119	255	117,6297	0,0115
17	125	380	128,8084	0,0305
18	153	577	146,4261	0,0430
19	159	263	118,3451	0,2557
20	171	807	166,995	0,0234
21	163	778	164,4015	0,0086
22	172	833	169,3202	0,0156
MMRE				0,0498
PRED(0,25)				0,91

4. Final remarks

In the paper we have addressed the problem of estimating the effort required to develop web applications, which represents an emerging issue in the field of web engineering [2,7,8,9,10,11,12]. In the context of traditional software systems, *Function Points (FP)* have achieved a wide acceptance to estimate the size of business systems and to indirectly predict the effort, cost and duration of their projects [1]. However, it is widely recognized that such method is no longer adequate for web-based systems, since it is not able to capture the specific features affecting the size and the effort required for those systems [8,9,11,12]. Nevertheless, the appealing features of the *FP* approach have motivated recent proposals of adaptation/extension of the method, meant to exploit its main ideas in order to predict the size of web applications. In particular, *Web Objects* represent an extension of *FP*, especially conceived for web systems [10,11,12]. It is characterized by the introduction of four new web-related components (multimedia files, web building blocks, scripts and links) added to the five traditional function types of *FP*. A different solution was outlined by Rollo, who employed *COSMIC-FFP*, an adaptation of *FP* originally defined for real-time applications, to measure functional size of an

Internet bank system [9]. Following his suggestion in [7] Mendes et al. provide a formal method which adopts *COSMIC-FFP* to measure size of static hypermedia web applications. In the paper we have extended their approach by considering dynamic web applications and defining a set of rules that allow us to measure functional size of client-server applications. Since *COSMIC-FFP* measure is focused on the counting of data movements, it turns out to be particularly suitable for client-server applications, which are characterized by large amounts of data movements. The proposed rules have been conceived to be applied on design documents, such as use cases and class diagrams in order to provide an early size estimation.

Several research directions can be planned as future work. First of all, further analysis is needed for the assessment of the method. Indeed, the empirical evaluation provided in the paper has to be considered a preliminary analysis, useful for encouraging us in further investigation. Data coming from the industrial world are presently being collected, in order to obtain more reliable results. Such data will be also used to perform a comparative analysis with respect to other proposals, such as *Web Objects*.

References

- [1] A.J. Albrecht, "Measuring Application Development Productivity", in *Proc. of the Joint SHARE/GUIDE/IBM Application Development Symposium*, Oct. 1979, pp. 83-92.
- [2] L. Baresi, S. Morasca, P. Paolini, "Estimating the Design Effort of Web Applications," in *Proc. of the 9th Intern. Software Metrics Symposium*, Sydney, 2003, 62-72.
- [3] J. Conallen, *Building Web Applications with UML*, Addison-Wesley Object Technology Series, 1999.
- [4] D. Conte, H.E. Dunsmore, V.Y. Shen, *Software engineering metrics and models*, The Benjamin/Cummings Publishing Company, Inc., 1986.
- [5] COSMIC: COSMIC-FFP Measurement manual, version 2.2, <http://www.cosmicon.com>, 2003.
- [6] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, Prentice-Hall, 1996
- [7] E. Mendes, N. Mosley, S. Counsell, "A Comparison of Length, Complexity & Functionality as Size Measures for Predicting Web Design and Authoring", in *Proc. of the EASE Conference*, 2001, Keele, UK, 1-14.
- [8] M. Morisio, I. Stamelos, V. Spahos and D. Romano, "Measuring Functionality and Productivity in Web-based applications: a Case Study", in *Proc. of the 6th Intern. Software Metrics Symposium*, 1999, 111-118.
- [9] T. Rollo, "Sizing E-Commerce", in *Proc. of the ACOSM 2000 - Australian Conference on Software Measurement*, Sydney, 2000.
- [10] D. Reifer, "Web-Development: Estimating Quick-Time-to-Market Software", *IEEE software*, vol. 17, no. 8, November/December 2000, 57-64.
- [11] M. Ruhe, R. Jeffery, I. Wiecezorek, "Using Web Objects for Estimating Software Development Effort for Web Applications", in *Proc. of the IEEE Software Metrics Symposium*, 2003.
- [12] M. Ruhe, R. Jeffery, I. Wiecezorek, "Cost estimation for web applications", in *Proc. of 25th International Conference on Software Engineering*, 3-10 May, 2003, 285 – 294.

Web based architecture for Dynamic eCollaborative work

Ioakim (Makis) Marmaridis, Jeewani Anupama Ginige, Athula Ginige
Advanced enterprise Information Management Systems Group
School of Computing and Information Technology
University of Western Sydney
{makis, achandra, athula}@cit.uws.edu.au

Abstract. Traditionally organisations have been collaborating to complement one another's capacity and capability. The nature of these collaborations has been long-term, taking a long time to establish, and once established stay in operation for a long period of time enabling fairly rigid IT infrastructure to be put in place to support these collaborations. Now organisations are beginning to collaborate on specific short-term projects that are put together in a very short period of time. A new IT infrastructure is needed to support this type of dynamic collaboration. It should facilitate the rapid flow of required information among the collaborating organisations to support the cross organisational workflow. Having analysed the requirements we developed a Web based architecture to support dynamic eCollaborative work. It is a peer-to-peer model and the information is exchanged among collaborating organisations based on workflow tasks. We also surveyed the technology required to implement this architecture and identified that two critical technologies need to be developed. We are in the process of developing these missing technologies and then we will deploy this architecture.

1. Introduction

Collaboration is defined as "to work together, especially in a joint intellectual effort" [1]. This is by no means a new term and it is not even directly relevant to information technology. Collaboration has been well known to happen many centuries before computers were invented, or even electricity for that matter.

Flowing from the definition of Collaboration above, eCollaboration, in today's business world is defined as "the use of internet based technologies to enable continuous automated exchange of information between suppliers, customers and intermediaries" Donnan (2002) [2]. To date eCollaboration is constantly gaining popularity and acceptance as a common business practice. The term eCollaboration takes on different meanings in

different contexts, such as eLearning, software development etc. This paper concentrates on how eCollaboration can be used to benefit organisations in today's highly competitive business world.

For two or more organisation to collaborate first there needs to be business level agreement between them as well as a degree of trust for each other. Typical example of such collaborative work would be for two plastic manufacturing companies; one specialising in plastic bottles and the other on caps for bottles, agreeing to jointly market and cross sell their complimentary products. In order for this collaboration to be realised, some of the internal business processes, such as planning and development, marketing etc., need to cross the organisational boundaries and be shared. By enabling this collaboration to happen over the Web, these companies can benefit in many ways [2]. In this paper we briefly look at the factors that impeded the eCollaboration once the higher-level business decision is made, followed by a discussion on the need for an eCollaborative framework and bring forward a Web based architecture based on Component Based EApplication Development (and Deployment) Shell (CBEADS[®]) [3, 4] that can support eCollaborative work.

2. What is Dynamic eCollaboration and the need for it

Although eCollaboration is gaining popularity as a common business practise, there is still some resistance in its uptake. Business level trust aside, there also needs to be considerable compatibility in several areas between the business partners; from their high level joint business processes to their exchanged documentation and all the way to their ICT infrastructure and systems [5]. Also, eCollaboration is not well suited for the rapidly changing business environment of today. Establishing the technical framework for eCollaborative work typically involves high setup and maintenance costs. As a result, the

collaboration links established between business partners are typically few and rigid.

Contrary to eCollaboration as discussed previously, Dynamic eCollaboration involves very flexible relationships between business partners. It facilitates common work on typically short to medium term projects. Its goal is to assist businesses in realising a true “sense and respond” approach to doing business.

Under Dynamic eCollaboration, new relationships can be created almost ad-hoc, and taken down just as fast, without tying the business partners into inflexible, long term relationships.

Admittedly, Dynamic eCollaborative work is not yet as widespread as one would expect, given the benefits it has to offer to the organisations practising it. To this also contributes the lack of the necessary ICT framework that can support such efforts [5].

2.1. The need for an eCollaborative framework

It is becoming obvious that ITC infrastructure will have to play an integral part of realising Dynamic eCollaboration. Robust technology is necessary to facilitate the rapid communication, exchange of documents, sharing of applications and inter application messaging that is needed in Dynamic eCollaboration.

Unfortunately, ICT has so far been hindering the process instead. To our knowledge, there is no single framework that companies can adopt and architecture they can follow in order to be ready for Dynamic eCollaboration, and this is where a change is needed.

If such a framework is put in place in an organisation, it will greatly assist its participation to eCollaborative efforts, both at the local, regional, national and even international level. A common framework and architecture for Dynamic eCollaboration will provide the host organisation with the benefits of rapid deployment of systems for internal communication, enhanced information flow between business partners and a consistent approach to managing the eCollaborative relationships.

Arguably, once an ICT framework with such capabilities is in place for an organisation, and the lower level infrastructure consistency and interoperability is ensured, the organisation can then shift its attention to actually doing what it does best, business.

3. Overview of the Conceptual Framework for Dynamic eCollaborative work

While keeping in mind the benefits a Dynamic eCollaborative framework has to offer, this section will proceed in describing such conceptual framework in more detail.

Such framework must be capable of supporting very flexible, almost ad-hoc establishment and tearing down of various collaborations. It should also be able to secure those collaborations by means of securing the associated communication channels and data exchanges.

In addition to that, the framework should drive down the costs and effort associated with creating, maintaining and tearing down eCollaborative projects. The changeover of eCollaborative links from dealing with a particular business partner to dealing with another one must be simple, quick and as inexpensive as possible. The framework must provide consistency in its approach to dealing with different eCollaborative projects, as well as managing the ongoing joint work relationship between organisations.

Security must be an integral part of the framework. It should offer a flexible mechanism for enforcing authentication and granular access control per eCollaborative project, for each business partner and the associated data that might be used or created during the joint work. Also, in order to further enhance security, the framework must not have a single point of control or failure that could risk the integrity of the entire eCollaborative network if compromised.

It is critical to ensure that if there is a security breach at a particular member organisation, this can be contained and by design will not put at risk the entire eCollaborative network.

Finally, given the importance that the framework for Dynamic eCollaboration will play in an organisation over time, it is critically important that it is robust by design and will maintain its availability. Also it is necessary to ensure that if the framework in one of the participating organisations stops working, this does not compromise the integrity or availability of the entire eCollaborative network.

4. Identifying Critical Technologies of the Dynamic eCollaborative framework and their availability

To build an eCollaborative framework that adheres to the requirements laid out in the previous section will involve many different technologies. In addition, we believe that not all of those technologies are readily available and that at least two of them would still need to be developed.

Table 1 below shows the relevant, critical technologies for implementing the Dynamic eCollaborative framework along with their availability today.

Table 1. Critical Technologies for implementing a Dynamic eCollaborative Framework

Critical Function	Suitable Technologies	Exists Today
Encrypting Communications between organisations	SSL / IPsec	Yes
Unified end-to-end Security across the entire framework	To be developed	No
Encryption of business documents exchanged	Cryptography (Asym. Key)	Yes
Auto configuration of connections between business partners and security policy management	To be developed	No
Non repudiation of Messages	Digital Signatures	Yes
Framework for developing and running eApplications	CBEADS® Web Sphere etc	Yes

The first of the two items that still need to be developed will provide an end-to-end security system, capable of creating, managing and enforcing policy-based directives across an entire eCollaborative project. Its main focus will be to provide a consistent virtual view of the resources and people on the project for the purposes of authentication and access control. Second will be to provide a mechanism for configuring the connections between business partners in an eCollaborative project with focus on the ease of establishing and tearing those connections down with minimum costs and expertise required.

5. Architecture of the eCollaborative framework

Flowing from the high level requirements of the conceptual framework for Dynamic eCollaborative work between business partners, this section describes a proposed implementation of such framework. We believe that the proposed implementation of the architecture can

effectively address all of those high level requirements for Dynamic eCollaboration.

We propose an implementation based on CBEADS®. CBEADS® was developed by the AeIMS research group at the University of Western Sydney, in Australia. CBEADS® offers a flexible component based environment [6] facilitating incremental development and deployment of the eBusiness applications that can be used to support Dynamic eCollaborative work. It is being actively developed for several years now offering a mature technology platform, which has now been deployed in few organisations.

As Figure 1 below depicts, we envisage that there should be a CBEADS® available to handle the interactions at each organisation that wishes to form or be part of a Dynamic eCollaborative network.

eCollaborative Network Formations

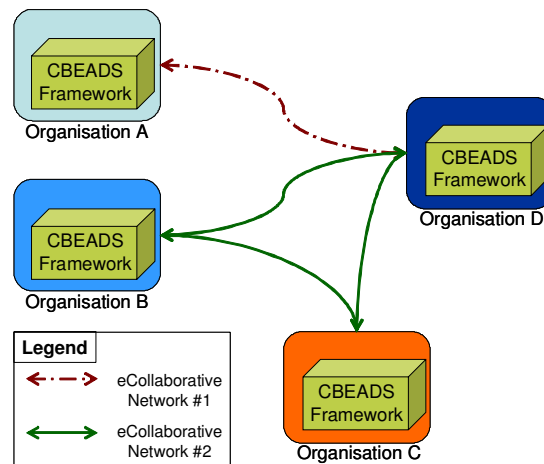


Figure 1 - eCollaborative Network Formations

With CBEADS® serving as part of the core technology platform, many of the issues around protocol interoperability and data interchange are alleviated. Also, because of the component-based approach adopted by CBEADS®, multiple applications can be present and running in any single organisation allowing multiple collaborations to go on in parallel with each other. At the same time, data and applications can be kept secure and segmented due to the built-in security features of CBEADS®.

Key strength of the proposed implementation is that it is based on the pure peer to peer architecture [7] of the conceptual framework while allowing for simple and inexpensive formation and evolution of the Dynamic eCollaborative networks.

Once again, the policy-driven security subsystem of CBEADS[®] can encapsulate and enforce decisions regarding what organisations should participate in what eCollaborative projects and also, what data and applications are to be shared, what users and groups of users are to access these and to what extend. In addition, to end an eCollaborative project it will be enough to revoke the respective policy through CBEADS[®] for all access to the associated data and application to be removed.

Figure 2 below shows the components of our proposed CBEADS[®] based implementation of the framework for Dynamic eCollaboration. The view is of a single participating organisation and amongst other also shows the relationship between the eCollaborative platform and other internal systems and data repositories.

Worth noting in figure 2 is the Access APIs layer that isolates, controls and monitors access to internal (legacy) resources from the outside. Also, the Security Subsystem that provides clearance for all access requests made to the organisation for the purposes of eCollaborating. As a result, all access to internal data and applications is centrally controlled and managed for the entire organisation through CBEADS[®].

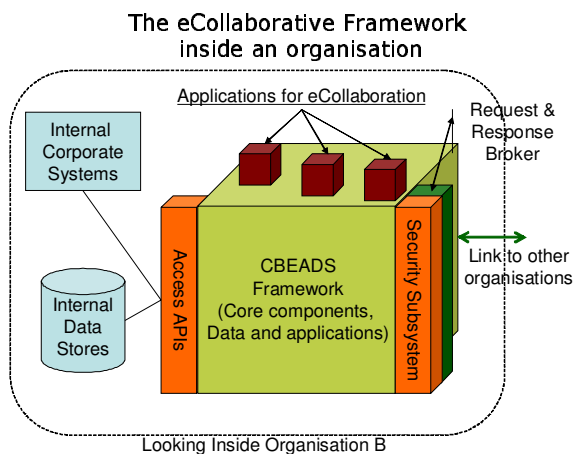


Figure 2 - The eCollaborative Framework inside an organisation

By taking this approach, any participating organisation is free to choose the level of data exchange it requires and can detach itself from the eCollaborative network at a later stage without having to recover data that has been published externally, as the case would have been in a shared portal establishment scenario [8]. Under this architecture, data remains in its native system (the organisation's data store) and only the elements needed for the eCollaboration are exchanged.

6. Conclusions and future work

We have presented an architecture that can facilitate Dynamic eCollaboration among organisations. We have validated this architecture against the detailed requirements such as no single point of failure, easy to set up and break down links to support the workflow activities related to the collaboration etc.

We also surveyed the technologies that can be used to implement this architecture. We have identified that before we can implement this architecture we need to develop two critical technologies. These are a consistent approach to managing the necessary policies as well as security across the entire framework and a mechanism for intelligent, automated configuration of the connections between business partners, as they are required. We are in the process of developing these technologies. Once developed, we plan to deploy a pilot of these systems to several organisations in the Western Sydney Region of Australia.

References

- [1] "The American Heritage[®] Dictionary of the English Language," vol. 2004, Fourth ed: Houghton Mifflin Company, 2000.
- [2] D. Donnan, "CEO/Presidents' Forum - Action Plan for Trading Partner e-collaboration," GMA CEO/Presidents' Forum June 7-10 2002.
- [3] A. Ginige, "New Paradigm for Developing Evolutionary Software to Support E-Business," in *Handbook of Software Engineering and Knowledge Engineering*, vol. 2, S. K. Chang, Ed.: World Scientific, 2002, pp. 711 725.
- [4] A. Ginige, "Re Engineering Software Development Process for eBusiness Application Development," presented at Fifteenth International Conference on Software Engineering and Knowledge Engineering, San Francisco Bay, USA, 2003.
- [5] K. Schuster, "Cross-Industry Standard Key to eCollaboration Success," in *News Release issued by Ticona*, Philadelphia, 2002.
- [6] I. Marmaridis, J. A. Ginige, A. Ginige, S. Arunatilaka, "Architecture for Evolving and Maintainable Web Information Systems," IRMA04, 2004.
- [7] V. Prasad and Y. Lee, "A scalable infrastructure for peer-to-peer networks using web service registries and intelligent peer locators," presented at Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on, 2003.
- [8] S. Woodman, G. Morgan, and S. Parkin, "Portal replication for Web application availability via SOAP," presented at Object-Oriented Real-Time Dependable Systems, 2003. (WORDS 2003). Proceedings of the Eighth International Workshop on, 2003.

Feature Value Propagation Analysis for Natural Language Grammars

Ettore Merlo¹, Michel Gagnon¹, Giuliano Antoniol², and Dominic Letarte¹

¹Department of Computer Engineering, École Polytechnique de Montréal,
P.O. Box 6079, Downtown Station, Montreal, Quebec, H3C 3A7, Canada
e-mail: {ettore.merlo, michel.gagnon, dominic.letarte}@polymtl.ca

²Department of Engineering, University of Sannio,
Research Centre on Software Technology (RCOST),
Via Traiano - Palazzo ex Poste - I-82100 Benevento - ITALY
e-mail: antoniol@ieee.org

Abstract

Grammars used in parsers for natural language are usually based on feature values that are propagated by the rules. In this paper, we present a flow analysis that has been developed for these grammars and we show that it is useful to identify defects in a grammar.

1 Introduction

The construction of a large-scale grammar for natural language parsing is a task that requires many men-years of work. Not only must we build the rules to cover all the possible syntactic forms, but we must also achieve a fine tuning of these rules to take into account all the subtleties of the language and the idiosyncratic forms. The task becomes even worse if our objective is a grammar tolerant to mistakes. Generally, this results in a big grammar and a large lexicon, both with many intricate features.

When one is faced with a situation that requires a modification of the grammar or the lexicon, it is very difficult to identify the impact of this modification in the parsing process. With the tools available at this moment, typically, we would resort to a pre-analysed corpus of sentences and check that the analysis of these sentences remains the same after the modification. The problem with this approach is that it is time-consuming and does not point specifically at the problem in the grammar design. For example, it would be useful to know what are the other grammar rules or lexical entries that could potentially be affected by some change in a grammar rule.

To assist this kind of impact measures, we propose to adapt a flow analysis that is well known in software engi-

neering. We will argue that using this simple method, we can easily identify problems in the design of a grammar.

In the next section, we present briefly the grammatical formalism that has been used to test our method. The flow analysis itself is formally described in section 3. Finally, a small experiment on a grammar for Portuguese is presented and used to show the advantages of our method.

2 Unification grammar

Unification grammar is a constraint-based formalism, where every symbol in the grammar rules is paired with a feature structure. Based on an unification process originally proposed by Kay [5] which is an extension of the Prolog unification, it is now used in the majority of mainstream grammar formalisms, such as Head-Driven Phrase Structure Grammar [9] and Tree-adjoining Grammar [10]. It is also at the base of the general purpose tools that are available for designing grammars, such as ALE [3, 2].

A feature structure is essentially a list of feature-value pairs. The structure is recursive in the sense that a feature value may be another feature structure, but here we will simplify by assuming that the value of a feature may be either an atom or a variable. This simplification turns easier the understanding of the propagation mechanism described in the next section, without affecting its theoretical foundations.

For example, the following structure says that feature a has the value 1, and that the values for features b and c are unknown but must be the same, since they share the same variable:

$$(F_1) \quad \begin{bmatrix} a & 1 \\ b & X \\ c & X \end{bmatrix}$$

The mechanism used to obtain the derivation tree is the unification. The algorithm is well known (see for example [4] for a description of the algorithm) and an efficient implementation is proposed in [6]. Intuitively, the unification algorithm tries to find a value for all the variables in such a way that the feature structures become identical. In the derivation process, a rule may be activated when its head feature structure may be unified with some feature structure in the body of another rule. The terminal nodes in the derivation are feature structures associated to lexical items. For example, the grammar illustrated in Figure 1 may be used to produce the derivation of Figure 2. Note that variables and atoms are expressed by capital symbols and uncapitalized tokens, respectively. Note also that the value of a feature may be propagated into another feature structure, in the body or the head of the grammar, by simply reusing the same variable.

$$\begin{aligned}
S &\rightarrow NP \left[\begin{smallmatrix} \text{NUM} & X \\ \text{VAL} & \end{smallmatrix} \right], VP \left[\begin{smallmatrix} \text{NUM} & X \\ \text{VAL} & \end{smallmatrix} \right] \\
NP \left[\begin{smallmatrix} \text{NUM} & X \\ \text{VAL} & \end{smallmatrix} \right] &\rightarrow DET \left[\begin{smallmatrix} \text{NUM} & X \\ \text{VAL} & \end{smallmatrix} \right], N \left[\begin{smallmatrix} \text{NUM} & X \\ \text{VAL} & \end{smallmatrix} \right] \\
VP \left[\begin{smallmatrix} \text{NUM} & X \\ \text{VAL} & \end{smallmatrix} \right] &\rightarrow V \left[\begin{smallmatrix} \text{NUM} & X \\ \text{VAL} & \text{intr} \end{smallmatrix} \right] \\
DET \left[\begin{smallmatrix} \text{NUM} & \text{sing} \\ \text{VAL} & \end{smallmatrix} \right] &\rightarrow \text{the} \\
N \left[\begin{smallmatrix} \text{NUM} & \text{sing} \\ \text{VAL} & \end{smallmatrix} \right] &\rightarrow \text{baby} \\
V \left[\begin{smallmatrix} \text{NUM} & \text{sing} \\ \text{VAL} & \text{intr} \end{smallmatrix} \right] &\rightarrow \text{slept}
\end{aligned}$$

Figure 1. Example of grammar

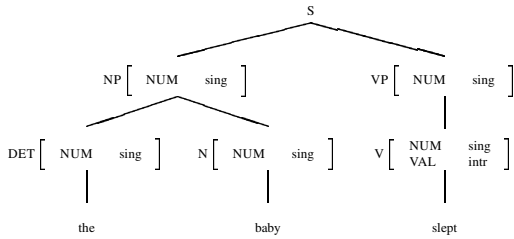


Figure 2. Example of derivation

3 Flow analysis

The feature value propagation can be fully described by giving the lattice of the problem, the partial order the lattice is defined on, the direction of analysis - forward or backward - and the flow equations. A general description of flow

analysis can be found in [1], while an approach for constant propagation analysis has been presented in [7].

A grammar G is defined as follows:

$$\begin{aligned}
G &= (N, T, RULES, S, FEAT) \\
propagSymbol &: RULES \rightarrow N \\
propagSet &: RULES \rightarrow \\
&\rightarrow FEAT \times \{VARS \cup ATOMS\} \\
unifySets &: RULES \rightarrow \\
&\rightarrow (FEAT \times \{VARS \cup ATOMS\})^n, (n \in \mathcal{N}^+) \\
vars &: RULES \rightarrow VARS
\end{aligned} \tag{1}$$

where N is the set of non-terminals, T is the set of tokens or lexical items, $RULES$ is the set of production rules used in the derivation process, S is the start symbol, and $FEAT$ is the set of features propagated by the grammar rules.

Some functions are defined on rules. *propagSymbol* returns the non-terminal on the left hand side of a production rule, *propagSet* returns the set of features and their values propagated by a rule, *unifySets* returns the n-tuple of sets of features and their values or variables used by the right hand side of a production rule for unification purposes, and *vars* returns the name space of variables in a rule.

The feature propagation flow analysis problem can be defined as follows: at any rule r in the grammar and for any feature a , determine the set V of values a may have.

A solution lattice for the flow analysis problem can be build by considering the partial order existing between sets of feature values. \perp represents the lattice node for which all features do not have any propagated value. \top represents the lattice node in which all the features in the grammar can have all the possible values in their domains. A generic node S_i in the lattice represents a particular configuration of information about the values of all the features in the grammar.

Suppose there are n distinct features named a_1 to a_n in a grammar G . Formally, a node S_i is denoted by the flow information associated to it as follows:

$$S_i = (V_{i,1}, \dots, V_{i,k}, \dots, V_{i,n})$$

where $V_{i,k} \subseteq \mathcal{P}(ATOMS)$ is the set of values associated with feature a_k .

The top and bottom of the lattice are respectively:

$$\begin{aligned}
\top &= (V_{\top}, \dots, V_{\top}, \dots, V_{\top}) \\
\perp &= (V_{\perp}, \dots, V_{\perp}, \dots, V_{\perp})
\end{aligned} \tag{2}$$

where V_{\top} is a special symbol indicating that feature a_k has an associated set of values which is equal to the total set of atoms and V_{\perp} is a special symbol indicating that feature a_k has an empty set of associated values.

For any node S_i in the lattice the partial order is defined as follows:

$$\begin{aligned} \perp &\preceq S_i \\ S_i &\preceq \top \end{aligned} \quad (3)$$

For any two nodes S_i and S_j in the lattice which are neither \top nor \perp the partial order between nodes is defined on the set inclusion between the sets of values corresponding to all the features in the grammar:

$$S_i \preceq S_j \iff V_{i,k} \subseteq V_{j,k}, 1 \leq k \leq n \quad (4)$$

Let's define $DG = (V_{DG}, E_{DG})$ to be the grammar derivation graph such that:

$$\begin{aligned} (V_{DG} &= N \cup T) \\ (v_1, v_2 \in V_{DG}), (v_1, v_2) \in E_{DG} &\iff \\ (\exists r_1, r_2 \in RULES \mid & \\ (v_1 \in LHS(r_1)) \wedge (v_2 = RHS(r_2))) & \end{aligned} \quad (5)$$

Flow analysis can be computed on the derivation graph. The direction of analysis is forward since we compute the current set of values based on previous values in the grammar derivation graph.

Feature propagation equations for any given grammar rule r are described in terms of the flow information coming in and out rule r . Sets $IN(r)$ and $OUT(r)$ denote such information and correspond also to nodes in the flow problem lattice.

Initially the flow analysis starts with with empty sets for all features, that is $\forall r \in RULES, IN(r) = OUT(r) = (V_{\perp}, \dots, V_{\perp}, \dots, V_{\perp}) = (\emptyset, \dots, \emptyset, \dots, \emptyset)$.

Let the flow information coming into rule r be $IN(r) = (V_{i,1}, \dots, V_{i,k}, \dots, V_{i,n})$ and the flow information coming out from r be $OUT(r) = (V_{j,1}, \dots, V_{j,k}, \dots, V_{j,n})$.

Elements of $OUT(r)$ can be computed as follows:

$$V_{j,k} = \begin{cases} V_{\perp} & \text{if } \nexists (a_k, v) \in propagSet(r) \\ \{v\} & \text{if } (a_k, v) \in propagSet(r) \wedge \\ & (v \in ATOMS) \\ \bigcup_{\substack{V_h \in IN(r) \mid \\ (\exists unifySet_i \in unifySets(r) \mid \\ (a_h, x) \in unifySet_i)}} V_h & \text{if } (a_k, x) \in propagSet(r) \wedge \\ & (x \in VARS(r)) \end{cases} \quad (6)$$

When several arcs of the grammar derivation graph merge in a rule, it is necessary to merge the flow information coming out from r_i and r_j , for example, to obtain the flow information coming into r_m . Let us assume that $OUT(r_i) = (V_{i,1}, \dots, V_{i,k}, \dots, V_{i,n})$ and $OUT(r_j) = (V_{j,1}, \dots, V_{j,k}, \dots, V_{j,n})$.

The merged information is:

$$\begin{aligned} IN(r_m) &= OUT(r_i) \bowtie OUT(r_j) = \\ &= (V_{m,1}, \dots, V_{m,k}, \dots, V_{m,n}) \end{aligned} \quad (7)$$

where \bowtie is the merge operator and

$$\forall k, (1 \leq k \leq n), V_{m,k} = V_{i,k} \cup V_{j,k} \quad (8)$$

Since the presented flow equations preserve the partial order defined by equations 3 and 4, fix-point solution is guaranteed to converge.

4 Experimentation and results

The grammar used in our experimentation is adapted from a Portuguese grammar that has been built for another project [8], where the objective was to test the sensibility of some parsers for phrase structure grammar. The grammar, which contains about 84 rules and uses a lexicon of about 7250 words, recognizes basic sentences. An important characteristic of this grammar is that it has been designed with the objective of making it insensible to common mistakes that could appear in texts written by Brazilians.

The flow analysis has been implemented in Perl on an AMD Athlon XP1700+ processor with 1477 MHz speed. The grammar derivation graph is obtained in 0.9 sec. and contains 100 nodes (84 for the rules and 16 for the terminal symbols). The grammar uses 24 features, 308 variables and 81 possible values for the features.

Table 1 gives a summary of the results. It gives the number of features in the grammar whose set of values has one of the following cardinality: maximal (all values are possible), any cardinality greater than one and less than maximal, singleton, and empty set.

Theses results show that in almost all cases, the method is not very informative about the possible values of the features. In 76% of the cases the domain value is unconstrained. This is not a surprise, considering the very conservative choice of union for the merging operator, which does not take into account the strong constraining effect of unification on the possible values. Even so, four singletons have been identified, and each one points to an actual problem in the grammar or a peculiar characteristic that is worth mentioning. In two cases, it happens that the value propagated to some feature in a rule is always the same, making this propagation useless. In this case, we can either remove the feature in every rule that propagates it, and replace the variable by the propagated value in the rule where the singleton has been detected. It may be also the case that some rule is missing that would propagated another value. In both cases, some decision must be made to fix the grammar, or at least document the idiosyncrasy.

In the other two cases of singleton, the feature has a unique possible value because all the entries in the lexicon instantiate this feature with the same value. The lexicon could contain other entries that would give another value to this feature, but at this moment it does not have such an entry.

Cardinality	Feature	%
Maximal	235	76
$1 < \text{card} < \text{Maximal}$	69	23
Singleton	4	1
Empty set	none	0

Table 1. Summary of results

5 Conclusion and future work

In this paper, we proposed a simple flow analysis that can be used to identify potential problems in the design of a grammar for natural language processing. As far as we know, this kind of technique, widely known in software engineering, has not been used in computational linguistic researches. We showed that even with a very conservative approach regarding the propagation, we can identify real problems in a grammar. Every instance of singleton or empty set points to a potential error in the grammar design.

We conclude that this approach is very promising, and should give more convincing results when applied to a very large grammar. Also, a refinement of the formalism, to make it reflect more precisely the effect of unification in the derivation process, should turn the analysis more expressive in terms of problems identified in the grammar design.

References

- [1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers—Principles, Techniques, and Tools*. Addison-Wesley Publishing Co., 1986.
- [2] B. Carpenter and G. Penn. Compiling types attribute-value logic grammars. In *Recent advances in Parsing Technology*. Kluwer, 1996.
- [3] Bob Carpenter and Gerald Penn. *The Attribute Logic Engine - User's guide*. Philosophy Department, Carnegie-Mellon University, 1994.
- [4] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.
- [5] Martin Kay. Functional grammar. In *Proceedings of the Fifth Meeting of the Berkeley Linguistics Society*, pages 142–158, Berkeley, 1979.
- [6] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, 1992.
- [7] E. Merlo, J.F. Girard, L. Hendren, and R. De Mori. Multi-valued constant propagation analysis for user interface reengineering.
- [8] Mariza Miola. Construção de gramática to português para um estudo comparativo da robustez de alguns algoritmos de análise grammatical. Technical report, Master Dissertation, Universidade Federal do Paraná, 2002.
- [9] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago, 1994.
- [10] K. Vijay-Shanker and A. Joshi. Feature structure based tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, pages 714–719, 1988.

Recovering Traceability Links between Requirement Artefacts: a Case Study

Andrea De Lucia, Fausto Fasano, Rita Francese, Rocco Oliveto

Dipartimento di Matematica e Informatica

Università degli Studi di Salerno

Via Ponte don Melillo, 84084, Fisciano (SA), Italy

adelucia@unisa.it, ffasano@unisa.it, francese@unisa.it, r.oliveto@tiscali.it

Abstract. *Recently, researchers have addressed the problem of recovering traceability links between code and documentation using information retrieval techniques [1], [11]. We present a case study of applying Latent Semantic Indexing to recovering traceability links between artefacts produced during the requirements phase of a software development process and discuss the application of our approach within an artefact management system.*

1. Introduction

Recently, researchers have addressed the problem of traceability link recovery between code and documentation mainly to help the software engineers in aligning them during maintenance [1], [11]. However, the problem of maintaining traceability links is not only restricted to source code and high level documentation, but involves all the artefacts produced during a software development process [2], [9]. This is especially true for evolutionary processes, where artefacts can be added, updated, or deleted in each phase, thus requiring a continuous reorganization of the traceability links.

Several research and commercial tools are available that support traceability between artefacts: TOOR [13], REMAP [15], and Rational RequisitePro [14] are only a few examples. Some tools [3], [4], [12] also combine the traceability layer and event-based notifications to make users aware of artefact modifications. For example, Cleland-Huang *et al.* [3] have developed EBT (Event Based Traceability), an approach based on a publish-subscribe mechanism between artefacts. However, the main drawback of these tools is the need for a manual detection and maintenance of the traceability links while the system changes and evolves, a difficult task for the software engineer [3], [4], [9]. Indeed, inadequate traceability is one of the main factors that contributes to project over-runs and failures [6], [10].

The aim of our work is to support the software engineer in the identification of the traceability links between software artefacts, throughout the development process,

by using Information Retrieval (IR) techniques [5], [8]. In particular, we present a case study of applying Latent Semantic Indexing (LSI) [5] to recover traceability links between artefacts produced during the requirements phase of a software development process.

IR techniques have already been used to recovering traceability links between code and documentation [1], [11], between requirements [9], and between maintenance requests and software documents [2]. In particular, our work presents similarity with work by Marcus and Maletic [11], concerning the use of LSI [5] as IR technique for traceability link recovery, although we apply it to different types of documents. Our work also presents similarity with work by Huffman Hayes *et al.* [9] concerning the application of traceability link recovery to artefacts produced during the requirement phase, although we use different type of artefacts, namely use cases and interaction diagram descriptions.

The paper is organized as follows. Sections 2 and 3 present the method and the case study, respectively. Section 4 concludes and discusses the application of our approach within an artefact management system.

2. Traceability Link Recovery Method

IR based traceability link recovery methods compare a document d_i (used as a query) against the other documents in the document space and ranks them according to their similarity with d_i . Moreover, these methods use some (fixed or variable) threshold to present the software engineer only a subset composed by the top documents in the ranked list having a similarity measure with d_i greater than or equal to the selected threshold. In this way, they restrict the document space, while recovering all the relevant documents.

Given a similarity measure between two documents, establishing if these have to be considered similar can be based on different approaches. A first method, called *cut point* [1], [11], consists of imposing a threshold on the number of recovered links regardless of the actual value of the similarity measure. In this way, we select the top μ

ranked documents for each query, where $\mu \in \{1, 2, \dots, n\}$. A different approach consists of using a threshold ε on the similarity measure. Among all the pairs of documents, only those having a similarity measure greater than or equal to ε will be retrieved. We have compared three methods to compute the cosine thresholds:

1. *Constant threshold*: this is the standard method used in literature [11]. The cosine threshold is constant.
2. *Variable threshold*: this is an extension of the previous approach. The constant threshold is projected in a particular interval, where the lower bound is the minimum similarity measure (instead of -1) and the upper bound is the max similarity measure (instead of +1). This has not been used in previous researches.
3. *Scale threshold*: a threshold ε is computed as the percentage of the best similarity value between two artefacts:

$$\varepsilon = c \text{ MaxSimilarity}$$

where $0 \leq c \leq 1$ [1]. Of course, the higher the value of the parameter c , the smaller the set of documents returned by a query.

It is worth noting that if *MaxSimilarity* is 1, the scale threshold and the constant threshold methods are equivalent. The scale threshold method is useful when the maximum similarity measure is low, while the variable threshold method is useful when the distance between the maximum and minimum similarity is low.

The IR method we used for traceability recovery is Latent Semantic Indexing (LSI) [5]. This is an important extension of the Vector Space Model [8] that assumes that there is some underlying or “latent structure” in word usage that is partially obscured by the variability in the choice of the words, and use statistical techniques to estimate this latent structure. A description of terms, documents, and user queries based on the underlying (“latent semantic”) structure is used for representing and retrieving information. In this way LSI partially overcomes some of the deficiencies of assuming independence of words, and provides a way of dealing with synonymy automatically without the need for a manually constructed thesaurus and preliminary text pre-processing and morphological analysis (stemming). Indeed, stemming is particularly challenging for languages, such as Italian, that presents a complex grammar, verbs with many conjugated variants, words with different meanings in different contexts, and irregular forms for plurals, adverbs, and adjectives [1].

The heart of LSI is Singular Value Decomposition (SVD), a technique closely related to eigenvector decomposition and factor analysis [5]. LSI applies this technique to the term-by-document matrix to decompose it into a set of k orthogonal factors from which the original matrix can be

approximated by linear combination. The result of SVD analysis is used to identify the similarity between each pair of artefacts. In fact, SVD provides the coordinates of the vector that represents an artefact; the similarity between a pair of artefacts is computed as the cosine of the angle between the corresponding vectors.

3. Case Study

We have experimented with the traceability link recovery method based on the LSI model on software artefacts produced during the requirement phase of a development project conducted by final year students at the University of Salerno, Italy. The project aimed at developing a software system implementing all the operations required to administer and manage a medical ambulatory. Table 1 shows the analyzed artefact statistics. The first two columns represent the type and the number of artefacts, respectively. The last two columns represent the average number of words composing an artefact and the average number of meaningful different words extracted from the artefacts, respectively.

Artefact type	# artefacts	Avg. # words	Avg # unique words
use cases	30	240	12
interaction diagrams	21	255	14
Total number	51	246.2	12.8

Table 1. Analyzed artefact statistics

The results of the application of LSI was assessed using two widely accepted IR metrics, namely, *recall* and *precision*. In general, for a given document d_i , the similarity measure and the defined threshold will be used to retrieve only the subset $retrieved_i$ of top documents in the ranked list that are deemed similar to d_i . The set of retrieved documents does not in general coincide with the set $correct_i$ of documents in the document space that are in fact similar to d_i . In general, the method will fail to retrieve some of the correct documents, while on the other hand it will also retrieve documents that are not correct. In our experiments the set $correct_i$ for each document d_i was provided by the original developers of the application.

Recall and precision for d_i can be defined as follows:

$$recall_i = \frac{|correct_i \cap retrieved_i|}{|correct_i|} \quad precision_i = \frac{|correct_i \cap retrieved_i|}{|retrieved_i|}$$

Both measures will have values between $[0, 1]$ (alternatively $[0\%, 100\%]$). If the recall is 1, it means that all the correct links are recovered, though there could be recovered links that are not correct. If the precision is 1, it means that all the recovered links are correct, though

there could be correct links that were not recovered. For the entire system the recall and precision are computed as follows:

$$recall = \frac{\sum_i |correct_i \cap retrieved_i|}{\sum_i |correct_i|} \quad precision = \frac{\sum_i |correct_i \cap retrieved_i|}{\sum_i |retrieved_i|}$$

In general, retrieving a lower number of documents for each query would result in higher precision, while a higher number of retrieved documents would increase the recall: in other word, the consequence of higher precision is a lower recall and vice versa.

In the first experiment we indexed all the artefacts within the same collection. Figure 1 shows the results. The 100% recall is reached with $\epsilon = 0.3$ for the constant threshold (with about 18% precision) and $\epsilon = -0.6$ (80% of the interval [min similarity, max similarity]) for the variable threshold (with about 20% precision). For the cut point method 30 artefacts are necessary to reach the 100% recall (with about 19% precision). The variable threshold performs generally better than the other two methods. It is worth noting that in this experiment we have not used the scale threshold, because for each query the maximum similarity measure was very high, thus giving similar results as the constant threshold. The best results were achieved with $\epsilon = 0.5$ for the constant threshold, with $\epsilon = -0.1$ for the variable threshold, and with 20 artefacts for the cut point method. For example, for the variable threshold we achieved a reasonable compromise between recall (about 85%) and precision (about 30%).

We observed that the artefacts belonging to the same category have similar structures and this increases their similarity measure, even if they are not relevant to each other (*false alarm* [7]). If a use case description is used as query, the related interaction diagram descriptions will have a lower similarity measure than irrelevant use case descriptions. To have more evidence from this, we performed a second experiment to recover traceability links between use cases and interaction diagrams. The results are shown in Figure 2: 100% recall is reached with $\epsilon = 0.28$ for the constant threshold and $\epsilon = 0.2$ (30% of the interval [min similarity, max similarity]) for the variable threshold. For the cut point method only 5 artefacts are necessary to reach the 100% recall. It is worth noting that for a variable threshold $\epsilon = 0.7$ we achieve more than 90% recall with 40% precision.

Due to this observation, we performed a third experiment, where the documents are indexed in two different collections, one for each category of artefacts. Queries are then performed against each document sub-space and for each of them a specific ranked list is created. Figure 3 shows the results: 100% recall is reached with $c = 0.4$ for the scale threshold and $\epsilon = -0.6$ (80% of the interval [min similarity, max similarity]) for the variable threshold. For the cut point method 27 documents in the use cases

collection and 7 in the interaction diagram collection are necessary to reach the 100% recall. In this case we did not use the constant threshold, because it does not take into account the differences in the maximum similarity values achieved in the two different collections of artefacts. It is worth noting that the results achieved in this case are better than the results achieved with a single collection of documents (compare Figures 1 and 3). In particular, for the variable threshold more than 40% precision with about 85% recall is achieved for $\epsilon = 0.3$.

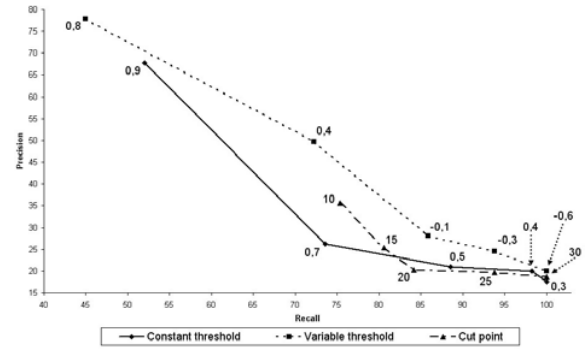


Figure 1. Precision/recall results in experiment 1

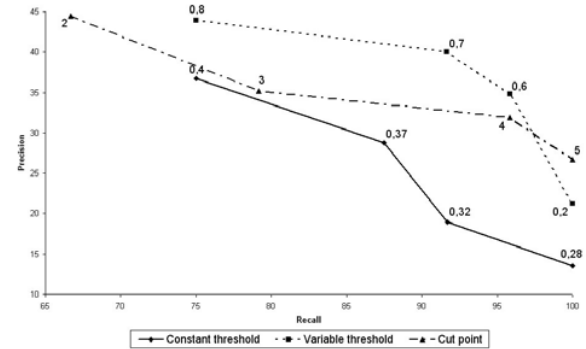


Figure 2. Precision/recall results in experiment 2

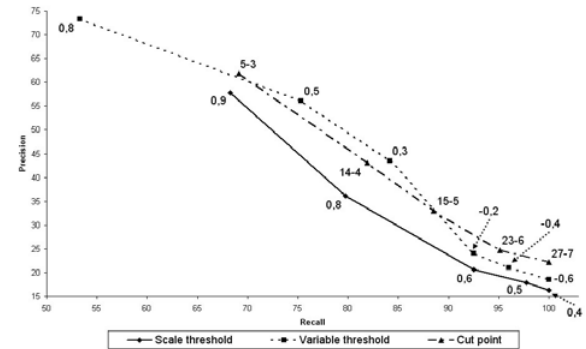


Figure 3. Precision/recall results in experiment 3

4. Conclusion and Future Work

We have presented a case study of applying a traceability link recovery method to software artefacts produced in the requirements phase of the software development process, in particular use case and interaction diagram descriptions. The contribution of our work can be summarized as follows:

- application of LSI [5] to requirements documents. Unlike other IR technique, LSI requires a slighter amount of pre-processing. To achieve comparable results, Huffman Hayes *et al.* [9] had to use an improvement of the vector space model based on a key-phase lists that has to be manually provided or extracted by introductory sections of a requirement document;
- use of a variable threshold to cut the ranked list of retrieved documents; in our case study this method seems to outperform previous methods [1], [11];
- categorization of artefacts of different types in different document subspaces to achieve better results. Our findings are also confirmed by other authors who used LSI on documents of different nature [7].

Ongoing experiments aim at extending these results to all the artefact types, including requirement, design, and testing documents, as well as code components. The first results show that the application of this technique to document spaces of larger sizes gives better results in terms of recall and precision, while keeping good performances.

We have implemented a tool that enables the software engineer to select the desired ranked list cut method and to tune the threshold. We plan to integrate this tool in ADAMS (ADvanced Artefact Management System), an artefact based process support system that integrates project management features and artefact management features, with particular emphasis on coordination of cooperative workers, context-awareness, and artefact versioning and traceability [4].

In particular, besides being useful for impact analysis during software evolution, traceability links in ADAMS are also useful to manage the software process and notify software engineers that the production of a given artefact can start, or that an artefact has to be changed, because of some changes in artefacts it depends on. Usually, software engineers are in charge of identifying traceability links between software artefacts, but as the project grows up, this task tends to be hard to manage [3], [4].

Another application of such a tool would be helping the software engineer in checking the loss of consistency in the usage of domain terms within software documents, in case a link is supposed to exist between two documents and it is not discovered by the traceability recovery tool.

References

- [1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation", *IEEE Transactions on Software Engineering*, vol. 28, no. 10, 2002, pp. 970-983.
- [2] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, "Identifying the Starting Impact Set of a Maintenance Request", *Proceedings of 4th European Conference on Software Maintenance and Reengineering*, Zurich, Switzerland, 2000, IEEE CS Press, pp. 227-230.
- [3] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-Based Traceability for Managing Evolutionary Change", *IEEE Transaction on Software Engineering*, vol. 29, no. 9, 2003, pp. 796-810.
- [4] A. De Lucia, F. Fasano, R. Francese, and G. Tortora, "ADAMS: an Artefact-based Process Support System", *Proceedings of International Conference on Software Engineering and Knowledge Engineering*, Banff, Canada, 2004 (to appear).
- [5] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by Latent Semantic Analysis", *Journal of the American Society for Information Science*, vol. 41, 1990, pp. 391-407.
- [6] R. Domges and K. Pohl, "Adapting Traceability Environments to Project Specific Needs", *Communications of the ACM*, vol. 41, no. 12, pp. 55-62, 1998.
- [7] S. T. Dumais, "LSI meets TREC: A status report", *D. Harman (Ed.) The First Text REtrieval Conference (TREC-1)*, NIST special publication 500-207, pp. 137-152.
- [8] D. Harman, "Ranking Algorithms", in *Information Retrieval: Data Structures and Algorithms*, 1992, pp. 363-392.
- [9] J. Huffman Hayes, A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval", *Proceedings of 11th IEEE International Requirements Engineering Conference (RE'03)*, Monterey, California, USA, 2003, IEEE CS Press, pp. 138-147.
- [10] D. Leffingwell, "Calculating Your Return on Investment from More Effective Requirements Management", *Rational Software Corporation*, 1997. Available online at <http://www.rational.com/products/whitepapers>.
- [11] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using Latent Semantic Indexing", in *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon, USA, 2003, pp. 125-135.
- [12] D. Nutter, S. Rank, and C. Boldyreff, "Architectural requirements for an Open Source Component and Artefact Repository System within GENESIS", in *Proceedings of the Open Source Software Development Workshop*, University of Newcastle (UK), 2002, pp. 176-196.
- [13] F.A.C. Pinhero and J.A. Goguen, "An Object-Oriented Tool for Tracing Requirements", *IEEE Software*, vol. 13, no. 2, 1996, pp. 52-64.
- [14] Rational RequisitePro, <http://www.rational.com/products/reqpro/index.jsp>
- [15] B. Ramesh and V. Dhar, "Supporting Systems Development Using Knowledge Captured During Requirements Engineering", *IEEE Transactions on Software Engineering*, vol. 9, no. 2, 1992, pp. 498-510.

Reengineering an Industrial Legacy Software Towards an Object-Oriented Knowledge-Based System^{*}

Hakim Lounis
Department of Computer Science
Université du Québec à Montréal
UQÀM, Canada
lounis.hakim@uqam.ca

Kaddour Boukerche
Centre de Recherche
Informatique de Montréal
CRIM, Canada
kaddour.boukerche@crim.ca

Houari A. Sahraoui
Department of Computer Science
and Operations Research
Université de Montréal, Canada
sahraouh@iro.umontreal.ca

Abstract. A software product is expected to fulfill some need and meet some acceptance standards that dictate the qualities it must have. This paper presents a reengineering work tending to increase to a significant degree some software qualities relevant in the management of production of a hydroelectric network. An object-oriented knowledge-based architecture is proposed to ensure an intelligent and automatic management of the knowledge in use in the daily decisional process of a major Canadian company.

1 Introduction

Reengineering is the examination and the modification of an existing system to reconstitute it in a new form and the subsequent implementation of the new form. The first phase of reengineering is some form of reverse engineering so as to abstract and understand the existing system. The second phase is traditional engineering or full restructuring using new specification and knowledge of the old system obtained from reverse engineering. This process is generally motivated by the will of moving old programs and systems to new platforms, as in source code translation, or restructuring programs that were corrupted by repeated maintaining activities. However, the most promising axis of reengineering is certainly moving legacy systems to emerging technologies and paradigms. Indeed, many organizations have been migrating their legacy systems to emerging technologies, e.g., Object-Oriented (OO) technology. Lehman and Belady present this migration as an economical choice through their three laws on the evolution of large systems [1].

OO approaches and languages have become quite popular, partially because of their potential benefits in terms of maintainability, reusability, separation of concerns, information hiding, etc. However, the vast majority of software available today is not OO. The effort to simply rewrite them from scratch using an OO approach would be prohibitive, and significant expertise recorded in the procedural software would be lost. The cost of manual conversion would also be prohibitive. Support coming from tools, documentation, and developers of the legacy software would ease the introduction of OO technology in many organizations. This kind of reengineering process could be especially helpful to integrate existing systems and new ones developed with OO approaches.

On the other hand, Knowledge-Based Systems (KBS) are used in numerous application domains, of which the field of hydroelectricity in which this work fits [2] [3] [4]. KBS are used to reproduce an expert's reasoning and are based on two distinct components: knowledge and reasoning. Separation between these two levels of intervention makes it possible to offer a flexibility of operation that many traditional software approaches are missing. KBS are presently an effective and useful solution to integrate the necessary analyses of hydropower experts and to meet the needs of the hydroelectric industry.

In the balance of this paper, we first present in section 2 the problem description. In section 3, we introduce what we consider as motivations for moving towards an OO knowledge-based architecture. In section 4, we describe the adopted solution and present its main features.

^{*}This work is part of an industrial project between the ALCAN Ltd group and CRIM, a research center. It was supported by a joint grant from ALCAN Ltd and NSERC, operation grant #CRDPJ 228746-99.

Finally in section 5, we conclude and present some of the lessons we learned.

2 Problem description

Alcan is one of the two world biggest player in the aluminum industry. With a total surface area of 73 800 km², the Alcan hydropower network under study, constitutes a territory larger than the province of New Brunswick (Canada). The network has, on average, an annual energy capacity of approximately 2000 megawatts; it includes 6 hydroelectric power stations, 28 reserve installations, 43 turbine-alternators groups (TAG), 850 kilometers of energy transport lines, a network of about thirty hydro-meteorological stations, etc.

The objective of planning the operation of such a network can be summarized as the satisfaction of the following requirements:

- Effective use of water
- Account of future hydrological uncertainty
- Satisfaction of energy need
- Respect of safety constraints.

To reach these goals, a semi-automated decision-making process of water stock management is used that consists of four steps:

- 1) Weather hydro measurements and gathering of the data
- 2) Data analysis
- 3) Weather and hydrological forecasting
- 4) Planning.

In these planning tasks, information processing systems based on mathematical models tested for this kind of applications, are used for optimization and simulation purposes. These models are implemented in Fortran within more than 65 routines totalising around 10.000 lines of code. A part of the legacy application contains what we consider as expert knowledge within its source code. However, most of the knowledge is used implicitly and in a non-automated way by Alcan analysts at different steps of their decision-making process.

This decision-making process is part of the knowledge management (KM) policy of Alcan Ltd. Currently, a lot of companies, often multinationals, face a significant problem of management of their knowledge, their know-how and their competences. They thus should constitute an alive and productive memory for their company, resting on three following main topics: the management of the experts and their expertise, the return of experiment, and the knowledge and information transfer in the company. From a theoretical point of view [5], KM makes it possible a company to manage (i) its

specific expertise, which characterize the company capacities in the study, the realization, the sale and the support of its products and its services and (ii) individual and collective know-how, which characterizes the company capacities of action, adaptation and evolution.

The knowledge used in the studied decision-making process, is many and varied. The problem is that this knowledge is often hidden in the code of the programs which use it, or consigned in internal documents, or even used implicitly by the experts, as in our case! This situation becomes more problematic, when the Alcan hydrological resources analysts wants to explore new scenarios, while modifying a little one of this knowledge. It has no other choice than to traverse the source code of the implemented programs, in order to make the discounted modifications there. It is, for example, the case of the operation rules of each power station and tank.

3 The Reengineering Motivations

The main disadvantage of the solution used since several years and described above, is the absence of separation between the knowledge level and the inference or reasoning one, in a product used in a knowledge intensive process! An immediate consequence is the restriction in the possibilities of investigation and exploration wanted by the Alcan analysts. Moreover, the adopted "black box" architecture produces a lack of flexibility of the whole decisional process. A consequence of that is the difficulty of maintaining and making evolve such a system.

An essential requirement of the KBS design process is the use of efficient representations of large amounts of knowledge; this ensures the consistency and effective exploitation of the KBS algorithms. The available knowledge can be explicit or implicit. An explicit representation consists of a symbolic expression of human expert knowledge. Rules are an example of that; they allow you to separate the expertise from the application code. This makes the application adaptable and maintainable. Since expert rules are externalized from the application code, they can be changed independently without recompiling the application. An implicit representation is knowledge that is usually hidden in data (numerical in our case). It requires further processing of the data before useful information can be extracted from it. For that, Machine-Learning (ML) techniques have been widely used to capture hidden knowledge from stored historical data. In each case, the goal was to determine trends or behaviour patterns that would allow the improvement of KBS procedures. For instance, ML techniques have been used in

hydroelectricity to produce rules from a power generation database [6] [7].

On the other hand, adding new functionalities is not the only goal of such a reengineering process. We also want to reach some quality attributes in the reengineered product. Some of these qualities are:

- **Evolvability:** software is evolvable if it allows changes that enable it to satisfy new requirements. It is a quality attribute close to flexibility. The initial design of a product as well as any succeeding changes must be done with evolvability in mind.
- **Usability:** a software system is usable if its human users find it easy to use. In our case, users are Alcan experts and analysts, and they have special needs.
- **Reusability:** this quality attribute is close to evolvability. It may be applied at different levels of granularity, from whole applications (including pieces of knowledge) to individual routines.
- **Understandability:** the activity of software maintenance is dominated by the subactivity of program understanding. Understandability helps in achieving many of other qualities, such as evolvability.

All these qualities could be in fact achieved thanks to internal software attributes, which deal largely with structure of the new software architecture.

4 The Object-Oriented Knowledge-Based Solution

As stated above, this work deals partly with the knowledge management of the Alcan experts, including data, thus allowing the hydropower resources planning or simulation. To help perform the planning tasks, we have developed a KBS called HYPERPIK (Hydro Power Resources Planning based on Inference and Knowledge). Figure 1 summarizes our system architecture. It consists of an inference engine that is coupled with a knowledge base resulting from the problem modeling. The knowledge base contains an explicit knowledge that is the symbolic expression of Alcan experts' know-how. A machine-learning framework exploits a historical database and produces explicit or implicit knowledge, depending on the selected learning mechanism [7]. The produced knowledge is then used in the decision process. In particular, it uses natural contributions flow values predicted from the historical database. These contributions flow values help evaluate the ability of the power system to face various contingencies and to propose appropriate remedial actions.

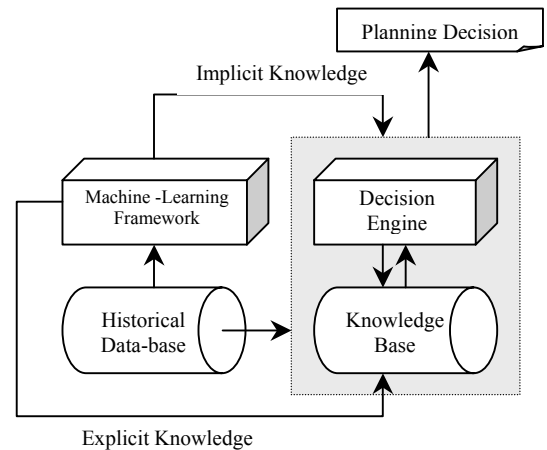


Figure 1. The HYPERPIK architecture

On the other hand, the planning step exploits explicit knowledge. It takes the form of rules we have built after weeks of an elicitation work with Alcan experts. The goal of this elicitation work was to explicit experts knowledge used implicitly in the previous semi-automated solution. Rule technology is based on the philosophy of providing fast and flexible software components to empower computer applications with “business” or “expert” rules capabilities. The general idea of a rule is that actions on the right-hand side are carried out whenever all the patterns on the left-hand side are successfully matched. A *pattern* is an expression that is capable of designating one or more objects. The objects result from our modelization of the hydropower domain and the classes diagram, given in figure 2, is an illustration of them.

The decision or inference engine processes the rules using the objects in a working memory. It implements a RETE algorithm [8] (it is widely recognized as by far the most efficient algorithm for the implementation of production systems) where rules are compiled into a network. Input data to the network consists of changes to working memory. Objects are inserted, removed and modified. The network processes these changes and produces a new set of rules to be fired. This process continues cyclically until there are no further rules to be fired.

The rules have a simple structure, composed of a header, a condition part and an action part. The header part defines the name of the rule, the packet to which the rule is attached, and its priority (if needed). The condition part utilizes the object-oriented structure of Java to carry out pattern matching on class instances, i.e. objects. This pattern matching binds (instantiates) variables to objects and field values. Rule conditions are also used to test

field values. This provides a filtering mechanism for objects. When the condition part of a rule is verified, i.e. valid objects have been found, the action part of the rule may be executed. Actions may vary from simple to complex, e.g. printing a message to creating new objects or calling a pre-existing Fortran routine (through a Java method).

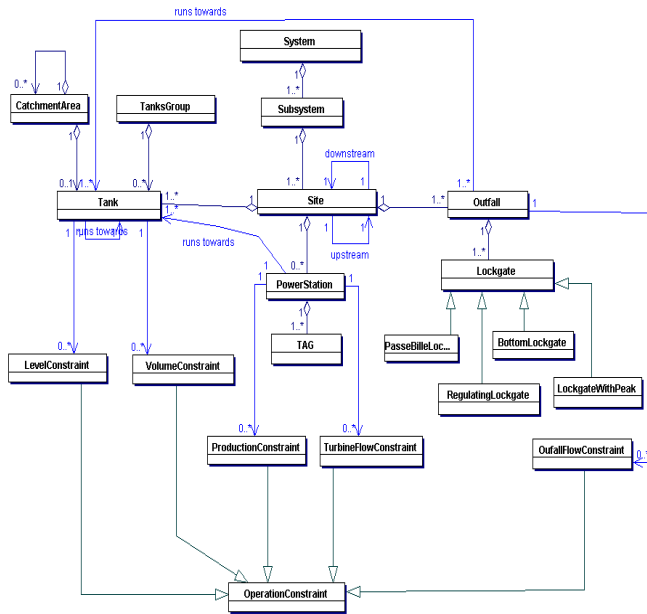


Figure 2. Objects involved in the planning process

The rules are written in the Ilog Jrules language¹ and the following one illustrate their structure:

```
rule short_TermDischargeRiskStJeanLake {
  packet = shortTermRiskStJeanLake;
  when {
    site(name == "StJeanLake");
    predicted_averageDischarge_InNextdays(currentDate + 7) > -100 ;
    predicted_averageDischarge_InNextdays(currentDate + 7) <= 0); }
  then {
    modify { shortTermRiskDischarge(currentDate) = 50; } } };
```

Our reengineering work yields to about 150 rules organized in 16 packets. A packet allows us to group rules with regard to their goal in the whole process. Examples of packets are: short-term risk at St-Jean Lake (see the rule above), overflow risk, Saguenay sub-system production, etc. This reengineering work yields also to a new interaction model between Alcan analysts

¹ Ilog Jrules is a general-purpose expert-system generator that combines rule-based techniques and object-oriented programming (www.ilog.com).

(final users of the system) and the system. Figure 3 gives the use-cases and illustrates these new functionalities and particularly the flexible way that the experts from now on have to configure their network or run a simulation session.

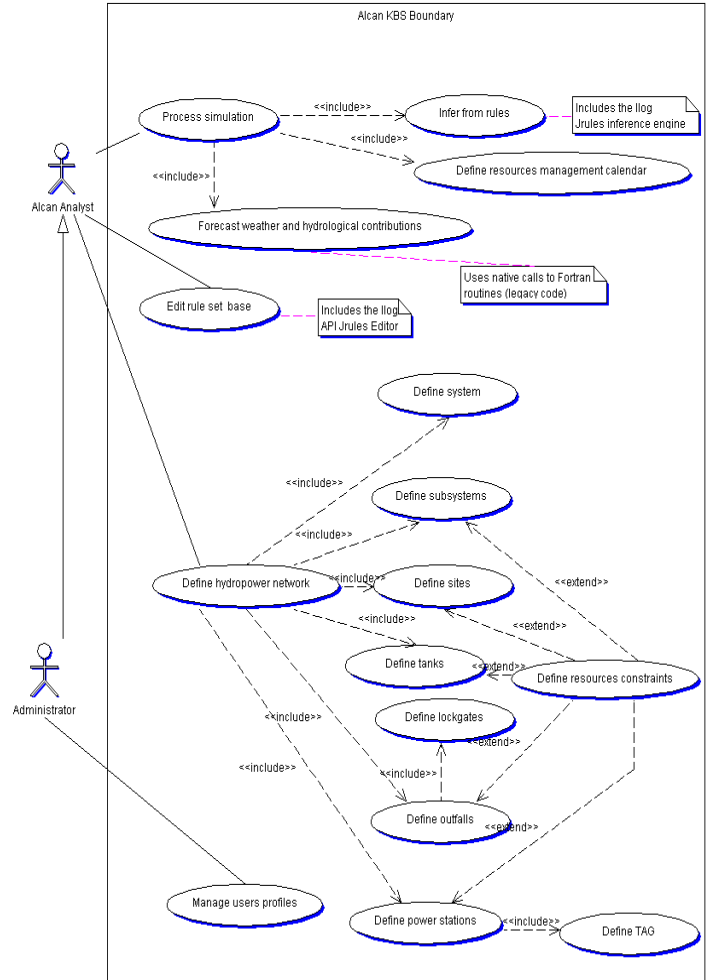


Figure 3. The Reengineered system use-cases

5 Conclusion and Lessons Learned

The reengineered system is currently used within the hydropower resources management group at Alcan. It results from a long collaborative process between authors of this paper and Alcan analysts. The latter were active and decisive actors; they have to maintain their Fortran routines (e.g., short term evaluation functions, water rise rate calculation functions, volume calculation functions, etc.) and we have to build a bridge between these functions and the objects methods we have implemented. The exercise was not so easy; we have to keep a good separation between what we consider as an expert knowledge and the procedures that exploit this

knowledge. This critical step of the project was iterative and it requires, even now, many adjustments.

The following points could summarize the strengths of the proposed solution:

- A greater flexibility of the tool during its use within the decisional process, by facilitating the exploration of new power network management scenarios. It is the main need of Alcan analysts.
- Better user interfaces allowing better usability during system configuration and simulation. For instance, the user-friendliness (i) of the configuration process of the network before each simulation (see use-case *<define hydropower network>* in figure 3), and (ii) of the updating process of the knowledge base (see use-case *<edit rule-set base>* in figure 3), are important factors.
- A better evolvability of the system thanks to the OO rules-based architecture. This architecture fosters evolvability, especially by offering the means of updating expert knowledge to explore new planning schemes.
- An understandability of the system much higher than it was in the previous system. Reaching understandability is a difficult task in complex systems with multiple functionalities. However, separating knowledge from the procedures that use it, and producing a new OO design are undeniable steps towards this objective.
- A better reusability of different components of the system. For instance, in our context, mathematical models implemented in Fortran reusable components are part of our global architecture.

This work is still in progress. We are working now on some extensions of the system. By adding priority factors to the rules, mainly to those dealing with management instructions, we expect to improve the whole performance of the tool. In fact, the tuning of such a system is a long and meticulous work; it is actually one of the main tasks of Alcan analysts. Finally, a next step will be to produce a rules verification module, in order to maintain the knowledge base free of anomalies (redundancy, inconsistency, etc.).

Acknowledgement. The authors want to thank Ms. Louise Rémillard and Janine Dufour from Alcan Ltd for their precious collaboration.

References

- [1] M. M. Lehman, L. A. Belady, "Program evolution", Academic Press, New York, 1985.
- [2] S. Samarasingh., A. McKinnon, J. Bright, "Expert System for Flood Management in Lake Manapouri", IEEE Comput. Soc. Press, Los Alamitos, CA, USA. First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, Dunedin, New Zealand, 1993.
- [3] J. Chang Tiao, D. Moore, "Reservoir Operation by the Use of an Expert System", Ohio Univ, Athens, OH, USA. Proc. of the 1994 ASCE National Conference on Hydraulic Engineering. Buffalo, NY, USA, 1994.
- [4] A. S. Leslie, A. Moyes, J. R. McDonald, G. M. Burt, J. McGowan, W. Charlesworth, "CEPE, Intelligent System for the Management of a Hydro-electric Scheme", Strathclyde Univ., Glasgow, UK. 31st Universities Power Engineering Conference, Iraklio, Greece, 1996.
- [5] M. Grundstein, "La capitalisation des connaissances de l'entreprise, système de production de connaissances", 1995.
- [6] M. Mejia-Lavalla, G. Rodriguez-Ortiz, "Obtaining expert system rules using data mining tools from a power generation database", Expert systems with applications, 14, 37-42. 1517, 1998.
- [7] H. Lounis, M. Boukadoum, V. Siveton, "Assessing Hydro Power System Relevant Variables: a Comparison Between a Neural Network and Different Machine Learning approaches", Proc. International Conference on Neuro-Fuzzy Technology (Neuro-Fuzzy2002), Havana (Cuba), 45-51, 2002.
- [8] C. L. Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem Artificial Intelligence", 19, 17-37, 1982.

Towards Knowledge Discovery in Software Repositories to Support Refactoring

Jörg Rech

Fraunhofer IESE

67661 Kaiserslautern, Germany

joerg.rech@iese.fraunhofer.de

Abstract Software repositories are typically used to store code together with additional information. These repositories are a valuable source to train knowledge discovery algorithms to detect code smells and other qualitative defects. In this paper we present a lightweight framework to detect previously unknown knowledge from software repositories to support refactoring. The results will be usable by software reengineers in the process of inspection and quality assessment of legacy systems.

1. Introduction

During the last years refactoring has become an important part in agile processes to improve the structure of software systems between development cycles. Especially in agile development under-engineering usually happens when the focus lies on adding more functionality to a system without improving its design along the way. When code works it is often simpler to engage the next task than cleaning up the previous work. Additionally, as systems are getting larger refactoring gets more and more complex and time consuming to do manually. Even if one knows how to refactor software it is not clear where and under what conditions what refactoring should be used [1].

In praxis, refactoring is a great challenge, as most software systems are badly implemented and therefore hard to evolve, maintain, and reengineer (e.g., Y2K). This aggravates if the software has to be optimized in order to meet new requirements, remove defects, or improve qualities like maintainability or reusability. Product managers need support to organize refactoring chains and to analyze the impact of changes due to refactorings on the software system. Analogously, quality managers and engineers need information to assess the software quality, identify potential problems, select feasible countermeasures and plan the refactoring process as well as preventive measures.

This paper describes a lightweight framework for the quality-driven, experience- and metrics-based instrument to support the refactoring of large-scale software systems. Developed instruments will give decision-support to software reengineers in the process of managing (i.e., measuring, monitoring, controlling, evaluating, and guiding) corrective, perfective, adaptive and preventive changes (esp., refactorings) to a software system. Based on the problems

described our framework is targeted to enable the monitoring and controlling of quality defects (a.k.a. “code smells”) in software systems based on software repositories (e.g., nightly, integration and release builds). The semi-automatic diagnosis of quality defects in a software system based on techniques from knowledge discovery in databases will help to detect refactoring candidates. Information from the diagnosis will support maintainers to select countermeasures (e.g., refactorings) and will act as a source for the initialization of preventive measures (e.g., code inspections). The evaluation of the work will be based upon information and source code from open source systems.

1.1. Related Work

Research in software maintenance has been undertaken in several large areas. As Bennett and Rajlich state in their roadmap paper the central research problem is the inability to change software easily and quickly [2]. Current research issues are to gain more empirical information about the nature of software maintenance, to build predictive models, to preserve and manage knowledge for the future maintenance of software, or the restructuring of code and data to remove unnecessary complexity [2, 3].

Previous research has resulted in behavior-preserving approaches to refactoring object-oriented software systems [4], tool support for refactoring application [5], methods for design-pattern based refactoring [6], metrics based and visually supported quality assurance with a similarity measure [7], modeling of object-oriented software to support later reengineering and refactoring [8], automated support for evolution and refactoring of object-oriented frameworks [9], and metrics based visual approaches to understand object-oriented software systems for reengineering [10]. Publications in the field of this thesis are concerned with collections of refactorings [1] as well as reengineering patterns [11].

Current research in the field of software refactoring is very active and is beginning to address formalisms, processes, methods, and tools to make refactoring more consistent, planable, scaleable, and flexible [12]. Metric based refactoring was currently only done for the refactorings “move method”, “move attribute”, “extract class”, and “inline class” based on one similarity measure with subsequent human interpretation [13].

2. Knowledge discovery and SW repositories

Today, several activities in software engineering like planning, monitoring, controlling, quality improvement, decision support, or automation benefit from knowledge engineering techniques [14]. Knowledge discovery in databases (KDD) is concerned with the detection of previously unknown information from large datasets. Discovery of knowledge is a process that can be divided into the five sub-processes Selection, Preprocessing, Mining, Validation, and Representation [15]. These sub-processes underpin the importance of clean data for the mining process (e.g., numerical without missing data) and the need for representation of clear valid knowledge (e.g., visualization of clusters).

Today, the term data mining is often used as a synonym for KDD. While Data Mining is the detection of “nuggets” in numerical data various forms of mining exists which examines different types of data. For example, text mining focuses on the extraction of knowledge from collections of long texts (e.g., books) while web mining focuses on typically small hypertexts (e.g., web pages), clickstreams, or log data.

The goals of KDD can be divided into the groups cluster discovery (i.e., answering “Are there related elements?”), class discovery (i.e., answering “How to classify elements?”), association discovery (i.e., answering “Do causal relations exist between elements?”), model discovery (i.e., answering “Do valid causal models exist?”), trend discovery (i.e., answering “What will happen in x days?”), pattern discovery (i.e., answering “Are there typical reoccurring structures (e.g., design patterns)?”), and correlation discovery (i.e., answering “Do correlations between (measured) variables exist?”).

Specific techniques for these goals like neural networks, decision trees, rough sets, or genetic algorithms can then, for example, be used to construct prediction models for decision support. These techniques as well as fuzzy set theory, case-based reasoning, or Bayesian analysis can be used to support software managers in the planning or controlling of their projects.

KDD promises to support various goals in software maintenance with the detection of knowledge in software repositories. For example, classification techniques can be used to detect similar methods or data structures in software systems. This can either happen on the code itself or on additional information (e.g., software metrics) attached to the code. Another example is the re-classification of methods from old classes into new ones to decrease coupling and increase cohesion of the renovated system. Furthermore, other scenarios are realistic like the automated classification of code fragments (i.e., methods or smaller) for the rapid development of code repositories to support reuse in agile environments.

2.1. Software repositories

Repositories in software engineering are used for nearly all elements, objects, or data related to software. After the speech of McIlroy late in the sixties repositories for code elements became more and more popular [16]. In the early eighties the experience factory (EF) – basically a repository about project experience and products – was established by Victor Basili [17]. Today, various other repositories for configuration management (e.g., CVS, SourceSafe), code reuse (e.g., ReDiscovery, InQuisiX), defect management (e.g., Bugzilla), or project databases exist in software engineering. Furthermore, if nightly- or daily-builds are compiled these also represent file-based repositories with valuable information about a project or software product. Software maintenance and development can benefit from these code repositories (i.e., CVS or nightly builds) if they are used to train defect detection techniques on previous builds of a software product.

In our repository source code from projects is measured and written into an XML document as well as a database for faster access. As depicted in Fig. 1 the code is cut into method blocks and contains metrics on every level (“MetricList”). The source code is attached to methods for later reuse and builds the basis for further diagnosis of defects or reporting.

```
<Package name="views">
<Class name="TableViewerExample.java"> {
  import org.eclipse.swt.SWT; ...
  private Table table; ... }
<Method name="main" modifiers="public"> {
  Shell shell = new Shell();
  shell.setText("Task List Example");
  ...
  tableViewerExample.run(shell); }
<MetricList name="Basic Method Metrics">
  <Metric name="LOC" value="7" /> ...
</MetricList>
</Method> ...
<MetricList name="Basic Class Metrics">
  <Metric name="NumberOfCasts" value="6"/> ...
</MetricList>
</Class> ...
<MetricList name="Basic Package Metrics">
  <Metric name="NumberOfClasses" value="6"/> ...
</MetricList>
</Package> ...
```

Fig. 1. Repository elements in XML

Comments in methods or javadoc elements are also attached and stored in special tags, but not shown in Fig. 1.

3. Discovery of quality defects in legacy software

Given the fact that activities in software product maintenance account for the majority of the cost in the software life-cycle [2] refactoring is a valid approach to prolong the software lifetime and improve its maintainability. Especially in evolutionary software development (i.e., agile methods) methods as well as tools to support refactoring become more and more important [12].

As shown in Fig. 2 we define six phases for the continuous discovery of quality defects. First we start with the definition of qualities that should be monitored and improved. This may result in different goals as, for example, reusability demands more flexibility or “openness” while maintainability requires more simplicity. Phase two represents the application area for KDD. It is concerned with the measurement and preprocessing of the software to build a basis for the defect discovery. Results from the discovery process (i.e., quality defects) can then be represented (e.g., visualized) and prioritized to plan the refactoring in phase three. Here the responsible manager or engineer has to decide which refactorings are to be executed in what configuration and sequence in order to minimize work (e.g., change conflicts) and maximize effect on the quality. In phase four the refactoring itself is executed on the software system by the (re-)engineers that results in an improved product. Phase five is used to compare the improved with the original product in order to detect changes and their impact on the remaining system. Finally, in the sixth phase we report the experiences and data about tasks, changes, and effects to learn from our work and continuously improve the model of relationship between quality, refactorings, and code smells.

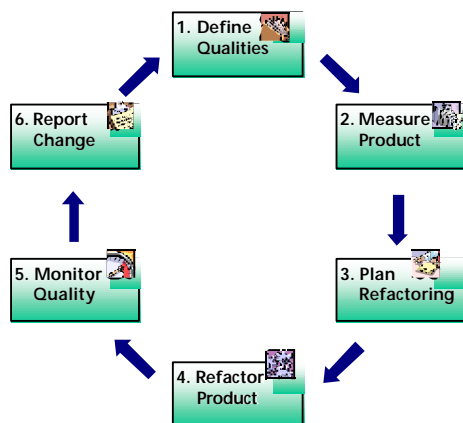


Fig. 2. Quality-driven metrics-based refactoring

As indicated in the previous paragraph the KDD subprocesses are grouped in phase two. We select source code from a specific build, preprocess the code and integrate the results into the software repository, analyze the data to detect quality defects, discover deviations from average behavior, cluster elements with severe or multiple defects, and represent or visualize discovered and prioritized quality defects.

For example, we use a classifier to classify if a method belongs to the class “defective methods” (i.e., if it is similar to methods that typically have quality defects). To train the classifier we use old source code and nightly builds to discover potential quality defects. The training code has to previously be analyzed by experts in order to detect and mark potential defects. The classification algo-

rithms can then determine under what attributes can be used by the classifier to distinguish defective from defect-free code. The trained classifier can then be used to discover quality defects in new software or current builds.

3.1. Current Status and Future Work

The current status can be described as work in progress. For the motivation and foundation of our work an extensive literature survey was made. Regarding the technology and tools a first prototype for the measurement of build sequences from software systems is currently in work. To evaluate our approach we will employ a quantitative analysis of open-source software systems, their versions, releases and nightly-build over a longer period of time. For the quantification of the problems in refactoring, evaluate the state of the art and praxis, and to get feedback about specific analysis results it is planned to integrate open-source communities.

To conquer the described problems and reach the chosen goals the following actions and ideas will be realized:

- Investigation of existing metrics as well as the development of new metrics to detect specific quality defects. Histories of nightly builds will be analyzed and used to accumulate massive amounts of data to detect changes and quality defects. Manual investigation of detected defects will help to evaluate the classifier and process. OSS communities will be informed about the analysis (i.e., they get a quality report) in order to receive feedback on the proposed changes and report structure. Investigated metric-defect dependencies will later support the goal-oriented planning of measurement activities with GQM.
- Investigation of existing refactoring as well as the development of new refactorings to increase specific qualities. Existing refactoring catalogs and quality models (e.g., ISO-9126) will be analyzed in order to synthesize a dependency model of refactorings mutually and between qualities and refactorings. Quality measurement plans from our projects or described in literature that are based on software product metrics will be analyzed and relations of metrics to qualities will be used to strengthen the dependency model. Pre-post evaluations of quality changes based on controlled refactoring experiments will be used to assess the impact of refactorings on different qualities. Investigated refactoring-quality dependencies will later support the goal-oriented planning of measurement activities with GQM.
- Elaboration of a quality-driven method to control refactoring processes and know where, when, and why to use what refactoring. A GQM-based process will be defined to support quality and product managers to reach a specific quality goal through refactoring. Furthermore, a CBR based repository of refactoring cases will be created in order to enable managers to reach different goals parallel based on the same metrics or to support him in what metrics he could include to reach other goals.

Optionally, we need to examine if software for specific application areas like embedded, distributed, or knowledge-based systems as well as product lines need additional techniques. In order to detect quality defects in the specifications of hardware (e.g., in VHDL) or knowledge (e.g., in OWL) new metrics might be needed.

Additionally, several side products will be produced like a correlation analysis of metrics to reduce measurement effort, the usage of metrics from different abstraction levels (e.g., requirements), the support of decisions in various phases (e.g., testing), or approaches for the visualization of quality defects in software systems.

4. Conclusion

The proposed framework promises the systematic and semi-automatic support of refactoring activities for product or quality managers. The incremental and low invasive (i.e., cheap) approach for the monitoring of software product quality in order to control refactoring activities will make maintenance activities more simple and increase overall software quality. Optionally, the project manager can use the monitoring of daily builds of the software to detect quality defects and initiate countermeasures during software development.

The framework developed expands the knowledge about quality and its measurement in software systems. It promises knowledge about how to detect quality defects (i.e., where should we refactor?) by software product metrics, the elicitation of knowledge about refactorings and their effect on software qualities (i.e., why should we refactor?), and if and in what sequence to refactor the software (i.e., when and how should we refactor?).

References

- [1] M. Fowler, *Refactoring: Improving the Design of Existing Code*, 1st ed: Addison-Wesley, 1999.
- [2] K. H. Bennett and V. T. Rajlich, "Software Maintenance and Evolution: A Roadmap," presented at Future of Software Engineering Track of 22nd ICSE, Limerick, Ireland, 2000.
- [3] H. Müller, J. Jahnke, D. Smith, M.-A. Storey, S. Tilley, and K. Wong, "Reverse Engineering: A Roadmap," presented at Future of Software Engineering Track of 22nd ICSE, Limerick, Ireland, 2000.
- [4] W. F. Opdyke, "Refactoring object-oriented frameworks," in *Graduate College*. Urbana, Illinois: University Illinois at Urbana-Champaign, 1992, pp. 142.
- [5] D. B. Roberts, "Practical Analysis for refactoring," in *Graduate College*: University of Illinois at Urbana-Champaign, 1999, pp. 127.
- [6] M. O. Cinneide, "Automated Application of Design Patterns: A Refactoring Approach," in *Department of Computer Science*. Dublin: Trinity College, 2000, pp. 231.
- [7] F. Simon, "Meßwertbasierte Qualitätssicherung: Ein generisches Distanzmaß zur Erweiterung bisheriger Softwareproduktmaße (in German)," in *Fakultät für Mathematik, Naturwissenschaften und Informatik*. Cottbus: Brandenburgische Technische Universität Cottbus, 2001, pp. 286.
- [8] S. Tichelaar, "Modeling Object-Oriented Software for Reverse Engineering and Refactoring," in *Institut für Informatik und angewandte Mathematik an der Philosophisch-naturwissenschaftlichen Fakultät*. Bern: Universität Bern, 2001, pp. 186.
- [9] T. Tourwe, "Automated Support for framework-based Software Evolution," in *Department Informatica*. Brussel: Vrije University Brussel, 2002, pp. 225.
- [10] M. Lanza, "Object-Oriented Reverse Engineering: Coarse-grained, fine-grained, and evolutionary software visualization," in *Institut für Informatik und angewandte Mathematik an der Philosophisch-naturwissenschaftlichen Fakultät*. Bern: Universität Bern, 2003, pp. 131.
- [11] S. Demeyer, S. Ducasse, and O. M. Nierstrasz, *Object-oriented reengineering patterns*. San Francisco: Morgan Kaufman Publishers, 2003.
- [12] T. Mens, S. Demeyer, B. Du Bois, H. Stenten, and P. Van Gorp, "Refactoring: Current Research and Future Trends," *Electronic Notes in Theoretical Computer Science*, vol. 82, pp. 17 pages, 2003.
- [13] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics based refactoring," presented at Fifth European Conference on Software Maintenance and Reengineering (CSMR), Los Alamitos, CA, USA, 2001.
- [14] L. C. Briand, "On the many ways Software Engineering can benefit from Knowledge Engineering," presented at Software Engineering Knowledge Engineering (SEKE), Ischia, Italy, 2002.
- [15] U. Fayyad, S. G. Piatetsky, and P. Smyth, "From data mining to knowledge discovery in databases," *AI Magazine*, vol. 17, pp. 37-54, 1996.
- [16] M. D. McIlroy, "Mass-produced software components," presented at NATO Conference in Software Engineering, Garmisch, Germany, 1968.
- [17] V. R. Basili, G. Caldiera, and H. D. Rombach, "Experience Factory," in *Encyclopedia of Software Engineering*, vol. 1, J. M. (ed.), Ed. New York: John Wiley & Sons, 1994, pp. 469-476.

Architecture for an Intelligent Web Application

Indra Seher, Athula Ginige

*School of Computing and Information Technology, College of Science, Technology and Environment,
University of Western Sydney, Australia.*

indra@cit.uws.edu.au, a.ginige@uws.edu.au

Abstract

There is very large amount of information on the Web today, and users of Web usually use search engines to find Web sites relevant to their queries. The outputs from search engines contain long list of documents including irrelevant ones. Using new Web technologies, we can develop applications that use an intelligent approach to find information on the Web. The output of these applications will be a page of synthesised information, rather than a list of Web site addresses. This paper proposes an architecture for such an intelligent application on the Web. When humans communicate, they can understand messages better, if they share the same context. We based our architecture on this concept and expand the user query with relevant contextual information of the user. Thus the architecture consists of a Communicator module to handle all interactions with the user, a Query Expander module to expand user query by adding contextual information, and a Synthesiser module to synthesise the output using existing structured data on the Web. There is also a module that stores information about the domains, based on domain ontology. This architecture was validated using some scenarios.

1. Introduction

There is very large amount of information on the Web today. According to Hobbes' Internet Timeline v7.0[3], the number of Web sites in December 2003 were about 45 million. The statistics of Online Computer Center Library [2] shows that there are 4,400 new web sites added every day. Users of Web usually use search engines to find Web sites relevant to their queries. Though search engines can rapidly process large number of Web documents, they do not consider the context of the documents. To process user query, user context in which the query is made has to be considered. Since search engines do not consider user context, they fail to identify the information requested by the user and produce long list of documents including irrelevant ones.

With emerging technologies of Semantic Web[1] such as XML, RDF, and Ontology, it is now possible for applications to understand and analyse document contents to aid automated processing. XML allows to structure data on the Web by separating the data from presentation information. Since XML lacks semantics, RDF is defined on top of XML to express meaning. As different data repositories could use different identifiers for the same concept, there should be a mechanism to identify relationships among terms. As concepts and relationships are specific to subject areas, their definitions will be specific to the area in consideration. A subject area or area of knowledge such as medicine, financial management and travelling is referred to as a domain. Ontologies allow to model a domain with computer-usable definitions of basic concepts in the domain and relationships among them. Ontologies will therefore enable some context-based access and interoperability across the Web.

With the availability of these upcoming technologies, the Web could have more intelligent applications to produce the needed information for the user. This could be a page of synthesised information rather than list of sites. This paper proposes an architecture for such an intelligent application on the Web.

2. Requirements of an Intelligent Web Application

This section describes the desired operations of an intelligent Web application. We used this as the basis to develop the architecture.

An intelligent Web application should accept user query and should synthesise a solution, which will be a single page of information, from existing data on the Web. User queries can be from different input devices from different locations. It may be from a mobile in a car, a telephone call from a workplace, a palm top at a work site, or from a Computer at home. Independent of the input device and media, user query has to be taken in by the application.

These could then be converted to text for processing.

User query alone may not be sufficient to process the request of the user. For example, a user query may be “Get me a place for Lunch”. To process this query, some more information about the user will be needed. These information may include user preferences such as restaurants and meal types preferred by the user. As these user preferences change less frequently, they could be kept stored by the application. To choose restaurants, the application has to consider information such as location of the user. This must be captured at the time of query. The application can use the input device of the user to capture environmental information such as location of the user and time of user query. Therefore, to process user query, the application has to expand user query using user context which could be user preferences and environmental information of the user.

After expanding user query, the application could begin to synthesise a response. For the synthesis, the application needs to use information on the Web related to user query. An information on the Web can be identified as related, if the domain of the Web site containing the information is same as the domain of the user query. Therefore, the intelligent application needs to identify the domain of user query and domains of Web sites.

After obtaining related information from the Web, the intelligent application can analyse these data to synthesise a result. As usage and naming of data could vary not only for domains but also for Web sites of the same domain, the intelligent application needs some knowledge about domains. This knowledge should include terms used in a domain and their relationships. Therefore, ontologies will play a crucial role in the representation of domain knowledge.

Using available data on the Web and domain knowledge, the intelligent application could synthesise a result for the user. This result may not be a single solution. When all required information is not specified, the output can have list of solutions out of which the user can select one. In the example given above, “Get me a place for lunch”, the output may not be a single meal from a single restaurant. If the user prefers a particular restaurant, still the output can be several types of meals at the restaurant. If the user is not particular about the restaurant, and therefore does not have the restaurant as part of his preferences or query, then the output can include list of restaurants, their locations, and available meals. If the user is not happy about the results, the user could give stricter preferences and the whole process could be redone.

These intelligent applications can also have learning

capabilities. Based on what user selects from the options provided, these applications could learn user preferences and use these in the future.

3. Proposed Architecture

Based on the above analysis, we have developed an architecture for an intelligent Web application. This architecture has four basic modules. They are Communicator, Query Expander, Synthesiser and Domain Knowledge Repository. Figure 1 shows the architecture and functions of each of the modules are explained below.

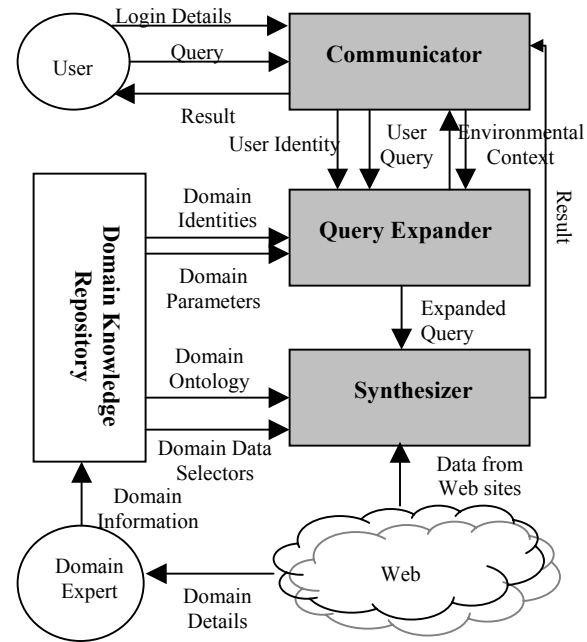


Figure 1. Proposed Architecture

3.1. Domain Knowledge Repository

The modules Query Expander, and the Synthesiser need some knowledge about the domain to expand the user query and to synthesise a result. Domain experts can analyse domains and structure the necessary knowledge which is stored in Domain Knowledge Repository.

3.2. Communicator

The Communicator handles all interactions with the user. It accepts user query in different forms depending on the user device. Independent of the methods used to capture query, the Communicator converts the query to text and sends it to the Query Expander. As the Query Expander needs to know the user of the query, the Communicator has to identify the user. This could be done by asking the

user to login or using more sophisticated biometric based approaches.

3.3. Query Expander

The Query Expander receives identification of the user and the query as text, from the Communicator. The Query Expander adds additional contextual information required to process the query. This contextual information will depend on the query. For example, for the query explained in section 2, “Get me a place for lunch”, user preferences such as user’s preferred meals, restaurants, types of meals and location of the user are needed to process user request. The Query Expander can store less frequently changing contextual information about the user. We refer to this information as user profile. In the above example, user profile should include user’s preferred meals, restaurants, and type of meals. The Query Expander can request the Communicator for changing user information related to the environment such as location of the user.

To support different scenarios and domains, the user profile should contain user preferences for the domains. Therefore, the Query Expander needs to identify the domain of user query, to decide the user preferences it has to add from user profile. Since it is hard for the applications to identify the domain of user query, a domain expert who has knowledge about the domain can analyse the domain, structure and store values that could be used to identify the domain. These values could be stored as part of the domain knowledge repository which we refer to as domain identities. Domain expert can also identify the contextual information needed for the expansion of queries in the domain. These also can be stored as part of the domain knowledge repository. We refer to these as domain parameters. For each domain we have a predefined template that specify the information required to process the query. The Query Expander expands the user query by filling the template using information from user query, user profile and the environmental context.

If the Query Expander cannot identify the user domain from the user query, then it could analyse the user query to identify keywords in the query. These keywords can be used to identify the domain using data on the Web related to the keywords. As the Query Expander does not have access to the Web, it sends these keywords to the Synthesiser, with a request to identify the domain. For example, for the query “Get me a ticket for SEKE2004”, the domain may vary depending on the location of the conference SEKE2004. If the location is beyond a distance the user can drive, then the domain may be “travelling”. When the domain is identified and sent by

the Synthesiser, the Query Expander can expand the query with user preferences stored in it, such as starting airport and preferred class to travel.

3.4. Synthesizer

Once the user query is expanded, the application can begin the synthesis. The synthesis is carried out by the Synthesiser module. The Synthesiser receives the expanded query which has user request and domain of request, from the Query Expander. Next, the Synthesiser has to decide the Web sites from where it could get data needed for synthesis. To choose the Web sites, the Synthesiser has to use some of the information in the expanded query. This information could vary for domains. For example, for the query “Get me a place for meal”, the Synthesiser could choose the Web sites of the restaurants that are closer to the user at the time of request. For the query “Get me a meal from McDonalds”, the Synthesiser has to choose data from Web sites of McDonalds restaurants closer to the user location. The information in the expanded query, that could be used to choose Web sites will be identified and stored by the domain expert as part of the domain knowledge repository. We refer to this as Domain Data Selectors. The Synthesiser could use these selectors to analyse the expanded query and could obtain relevant Web sites. Web data in the sites are assumed to be available in XML format.

The Synthesiser needs to understand the information obtained from Web sites, to synthesise a solution. Different Web documents could use different terms with the same meaning within the same domain. Therefore, there will be a need to model domains by defining the terms used in the domains and their relationships. As ontologies allow these definitions, domain ontology could be the main component of the domain knowledge repository. Initially, the domain ontology can be defined manually by the domain expert.

With the aid of domain ontology, the synthesiser will next synthesise a result using information obtained from Web sites related to the domain. The result may not contain a single solution. The number of solutions will depend on the amount of expansion done to the query. In other words, it will depend on the extent to which the user request was understood. Stricter preferences of the user may lead to more expansion to the query, giving less number of solutions or just one solution. When there are multiple solutions, the synthesiser could rank the solutions based on some of the domain data, which are also identified by the domain expert and stored as part of the domain knowledge repository.

The Synthesiser will send the result to the Communicator to present it to the user. When the result has more than one solution, user can choose the one user prefers. If the user is not satisfied with the result, user can add further restrictions or preferences to the query. Then the whole process will be redone. In these situations, the intelligent application could learn more about user preferences, and could update stored user profile.

4. Validating the Architecture

We used several scenarios with different text inputs in English to validate this architecture. Out of these, one scenario of the domain “meal”, is explained below.

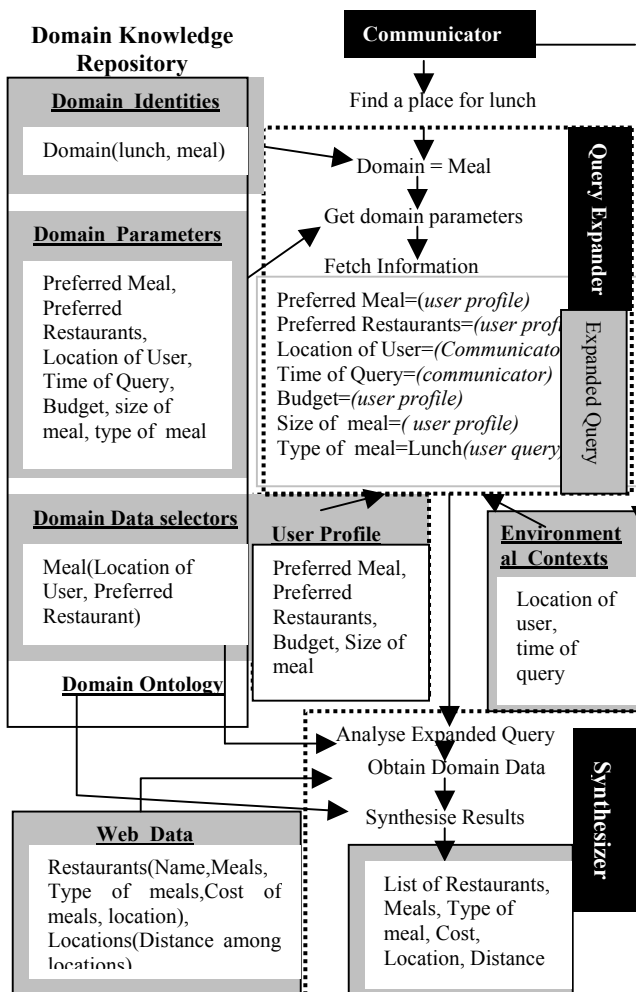


Figure 2. Information flow and activities of a Scenario

The text form of user query is “Find a place for lunch”. Using stored domain identities and the word “lunch” in the query, the Query Expander can identify the domain as “meal”. Next, the Query Expander will use the domain knowledge repository and obtain domain parameters that need to be provided. For this example, the parameters are

preferred meals, preferred restaurants, location of the user etc. Domain parameters indicate information stored in the Query Expander. The Query Expander could get most of this information from the user query and user profile. It gets user location specific information from the communicator. This information is next sent to the Synthesiser. The Synthesiser analyses the expanded query using stored domain data selectors to choose Web sites from which it needs to get information. In this example, it will make use of location of user, and preferred restaurants to choose Web sites. The result will contain list of restaurants, meals, types of meals, cost, location and distance to travel.

5. Conclusion and Challenges

Though we found this architecture to be domain independent and suitable for several scenarios, some of the issues given in the following paragraph have to be addressed before a system like this can be widely used.

As domains to which queries belong are important for synthesis, domains have to be correctly identified. For proper identification, there should be an agreement in naming domains. Still identification may be harder for queries which do not contain any key words related to stored domain identities. Out of the available Web sites, the sites related to the requested domain have to be identified correctly. Meta tags available in the sites can be used for this purpose. If the user query is accepted in a form other than text, it has to be converted to text prior to the expansion. This will involve several complex tasks depending on the form of inputs such as voice. Using the devices used by the user, the system has to capture user context such as location, time etc. The main component needed for synthesis is the domain ontology. This needs a proper study of heterogeneous data sources on the Web and their representation. Information on the Web sites change with time. Thus, proper maintenance of domain ontology is also needed.

References

- [1] Tim Berners-Lee, J. H. a. O. L. (May 2001). "The Semantic Web." Scientific American(May 2001). <http://www.scientificamerican.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21&catID=2>
- [2] Online Computer Library Center, Web Characterization Project, "Number of web sites." http://atlantis.uoc.gr/doc/Part%20II_web_site_numbers.doc
- [3] Robert H'obbes' Zakon, "Hobbes' Internet Timeline v7.0", <http://www.zakon.org/robert/internet/timeline/>
- [4] Peter Clark, "Some Ongoing KBS/Ontology Projects and Groups", <http://www.cs.utexas.edu/users/mfkb/related.html>
- [5] Hans-Georg Stork, "Semantic Web Technologies in Europe's IST Programme 1998 – 2002", http://www.ercim.org/publication/Ercim_News/enw51/stork

DMTF - CIM to OWL: A Case Study in Ontology Conversion

Dennis Heimbigner

Computer Science Department
University of Colorado
Boulder, CO 80309-0430 USA
dennis.heimbigner@colorado.edu

Abstract

The process and problems associated with translating a specific software engineering ontology into a different ontology language are described. The software engineering ontology is the Distributed Management Task Force (DMTF) Common Information Model (CIM). The target ontology language is the W3C OWL DL language. The specific translations are described for various CIM constructs. Difficulties in translation are characterized.

1. Introduction

As part of a larger project, we had a need to integrate a number of software engineering models. Our approach was to choose a common modeling language into which all of the other models would be translated. Once translated, the various models could then be integrated.

Each of the existing models used a different modeling language, which meant that we either had to choose one of them or choose a language different from all of the others. After some examination, it became clear that each of the existing modeling languages was closely tied to the model's content, and thus not suitable for representing the other models. Further, each of the modeling languages appeared to be ad-hoc in one or more parts.

In light of this, we chose to use a completely separate modeling language, namely OWL [1] which is the latest ontology language from the World Wide Web Consortium (W3C). OWL had well-defined semantics, and was intended to represent the content of a wide variety of web-based data.

Our initial goal was to translate the Common Information Model (CIM) developed by the Distributed Management Task Force (DMTF) [2]. This model is designed to represent "...an implementation-neutral schema for describing overall management information in a network/enterprise environment".

To be precise, CIM is a model of the domain of managed software systems. Thus it has notions such as services, clients, and software components. This domain is represented using a more-or-less classical object-oriented programming model. For clarity, the modeling language will be referred to as CIM-O, and the domain representation (specific uses of CIM-O) will be referred to

as CIM-D. The term CIM will be used for the combination. Our goal, then, was to re-represent the CIM domain (CIM-D) in OWL. The approach taken was to translate the CIM-O constructs used in the CIM-D representation to corresponding OWL constructs.

To give some additional background, the goal of the overall project is to address the problem of detecting previously unseen intrusions (e.g., viruses and worms) using *structural anomaly* detection. Instead of checking the behavior of software systems looking for anomalous behavior, our approach validates the structure of a running system against a comprehensive model describing that structure. Deviations are used to signal possible intrusions. This project requires a relatively complete model of the structure of a software system at run-time. This in turn requires the integration of a number of existing models, none of which alone is sufficient.

2. CIM-O Overview

The CIM-O language is a typical, although somewhat ad-hoc, object-oriented language. It has notions of class, attributes, and methods. The attributes represent named values associated with instances of the classes.

2.1. Classes

Classes in CIM-O use a single-inheritance model, so a basic class has the following skeleton

class <classname> : <superclass> {...}

where the superclass specification is optional. The body of the class is enclosed in curly brackets and contains a sequence of attributes and methods (with arguments).

2.2. Attributes

Attributes take one of the following general forms:

<simple type> name;

or

<class name> REF name;

The first case is used for a fixed set of literal types such as string, uint16 (e.g., 16 bit unsigned integer), Boolean, and so on. The second case is used when the attribute value is a pointer ("REF") to an instance of some class.

Any attribute may be designated as having a vector of values. Thus one might say

```
string commands [];
```

to indicate that a command attribute is a vector of strings.

Alternatively, one may indicate that an attribute has an initial/default value using this syntax.

```
uint16 priority := 0;
```

2.3. Methods

A method (procedure) has the following general form

```
<return type> name (arg, arg, ...);
```

where each argument has the form of an attribute.

3. OWL Overview

We assume familiarity with OWL [1], or at least RDF [3]. OWL is basically much like the semantic models that were popular in the early 80's in the AI and Database communities. It has notions of class and property. In fact, it is essentially constructed using only binary relations consisting of a relation name, a domain set and a range set (a triple in RDF terminology). Thus, subclass inheritance is a relation between the parent and subclass.

Note that OWL comes in three "flavors": lite, DL, and full. The term DL stands for "description logics" because this level of OWL is equivalent to description logics in expressive power. We adhere to OWL DL except where noted.

4. Translation Phase 1

The first translation phase addressed the most significant problems: how to translate the specific classes and simple attributes used in CIM-D.

4.1. Class Conversion

Converting a specific class skeleton in CIM-D to OWL is relatively straightforward. The following class

```
class Service : System {...}
```

would translate to the following.

```
<OWL:Class rdf:ID="Service">
```

```
<  rdfs:subClassOf rdf:resource="#System"/>
```

```
</OWL:Class>
```

OWL supports a variety of class formation mechanisms, including multiple inheritance, union, and intersection. But these are unneeded for translating CIM-D classes, although they are needed for translating attributes.

4.2. Attribute Conversion

At first glance, it might appear that CIM-O attributes can naturally be converted to OWL properties. Unfortunately, a number of conversion problems surfaced.

The first problem, name scoping, is one that occurs repeatedly in any attempt to map CIM-O to OWL. In

CIM-O the domain of an attribute is implicitly scoped to the class in which the attribute is defined. One would normally map an O-O attribute to an OWL property with the class being the domain and the value type of the attribute being mapped to the range of the OWL property. We would expect to map the specific CIM-D attribute

```
string Name;
```

to the following OWL format.

```
<OWL:DataTypeProperty rdf:ID="Name">
```

```
<  rdfs:domain rdf:resource="#Service"/>
```

```
<  rdfs:range rdf:resource="xsd:string"/>
```

```
</OWL:DataTypeProperty>
```

This is technically correct, but incomplete, because in OWL, normal properties have global scope, and hence are independent of classes. If, as is common, the same CIM-D property occurred in another class, then this would conflict with the existing definition of the Name property because we would be trying to define the property with two different domains.

A similar problem exists for the range specification. The default is that the range of a property is global to all uses of the property. Thus we cannot have an alternative definition of Name whose range is integer instead of string.

There are basically two ways out of these problems.

1. We can rename the property to include the domain class and thus make each property unique. Thus, the above example would be converted from Name to something like Service_Name. This also implicitly handles the range problem because each unique property can have whatever range it requires.
2. We can utilize the OWL *allValuesFrom* restriction on the property to essentially indicate that when the domain comes from a specific class, the range is the class specified by the *allValuesFrom* restriction.

We chose the second solution, but recognized that it complicated our translation process. The first solution was rejected because it would be difficult to decide the name of the property to use when constructing instances, and it apparently would complicate the handling of inheritance. In choosing solution 2, we hoped it would not be needed very often. In many cases, attributes were in fact unique to a specific class; hence, they could be defined using the simple approach.

In some cases, all definitions of attributes with the same name in fact had the same range. This is true of the Name attribute; its range is always a string. This would result in the very general definition such as the following.

```
<OWL:DataTypeProperty rdf:ID="Name">
```

```
<  rdfs:domain rdf:resource="OWL:Thing"/>
```

```
<  rdfs:range rdf:resource="xsd:string"/>
```

```
</OWL:DataTypeProperty>
```

This case can be extended because even if not all ranges are the same, most of them may be the same, in which case the more complex use of a restriction need only be used for those exceptional situations. If it turned

out that Name had an integer range (sint32) for the class Device, then we would add a specific restriction for that class.

4.3. Attribute Objectification

CIM-D contains one specific class whose purpose is to represent directed dependency relationships between two arbitrary objects. This class, named “CIM_Dependency” has two attributes: *Antecedent* and *Dependent*. The direction is from *Antecedent* to *Dependent*. Subclasses of this class are frequently created in CIM-D to represent specific dependency relationships. The following example defines a relationship between a service object (CIM_Service) and a managed element, which is any generic software object.

```
class CIM_ProvidesServiceToElement
: CIM_Dependency {
  CIM_Service REF Antecedent;
  CIM_ManagedElement REF Dependent;
};
```

Such classes are often referred to as *objectified* attributes. Clearly any such relationship could be represented by an attribute attached to the antecedent class, but instead, it has been converted to a separate class of objects representing the relationship. Objectification is typically used when either the relationship does not naturally associate with the antecedent (or the dependent, for that matter), or when the objectified relationship is actually an n-ary relationship for $n > 2$. CIM-D uses objectification for both reasons. Many of the CIM_Dependency subclasses are independent of the antecedent, and as well, some of the subclasses have additional attributes that convert them to ternary relationships.

We have followed the CIM-D model in this approach and have kept objectified relationships in our translation. Of course, for n-ary relationships, we have no choice since OWL does not support n-ary properties.

5. Secondary Translation Issues

The first phase of translation indicated that the process was generally feasible. The second phase involved examination of various secondary features of CIM-O used in CIM-D to see how they might be translated to OWL.

5.1. Vectors

CIM-O class attributes can specify that their range is actually a vector of values (Section 2.2). Mapping vectors into OWL is one of those topics for which no discussion could be found in the OWL documentation. Frankly we have no idea what the “proper” mapping should be, so we have developed our own representation using the `rdf:List` class which is defined as OWL’s sole container type; this is at the technical cost giving up the random access behavior of the vector. OWL uses only lists because they contain a fixed number of elements: the *first* and *rest*

properties as of the `rdf:List` class. This two-element definition apparently simplifies reasoning about lists.

Our solution defines for each class C, another class, `vectorC`, representing the vectors whose elements are of type C. The `vectorC` object is defined as follows.

```
<OWL:Class rdf:ID="vectorC">
  <rdfs:subClassOf rdf:resource="rdf:List"/>
</OWL:Class>
```

This definition is unsatisfying because it places no restrictions on the types of elements in the list. In an attempt to remedy this, we make use of the *allValuesFrom* restriction to force the values to be of the correct type and to force the list nodes to be of type `vectorC`.

A possible solution recently suggested was to model vectors as multi-valued properties and use the *MaxCardinality* restriction to handle non-vector attributes. This would, however, appear to give up the ordering property of a vector, which the list approach keeps.

5.2. Default Values

Another feature of CIM-O is the assignment of default values for attributes. Surprisingly, modeling default values turns out to be very difficult in OWL. The subject has been extensively discussed in the W3C. From that discussion we concluded that (1) defaulting cannot be included in the standard OWL DL model, and (2) there was no agreement about defaulting and users were encouraged to experiment. In effect, we were on our own. This was irritating, and almost made us abandon our choice of OWL as our common modeling language.

The solution we are currently pursuing is to treat defaulting as a form of inference. This idea came from experiments with the Jena ontology database system [5]. Figure 1 shows one view of the architecture of Jena. The idea is that one has a base graph representing some model — our converted CIM-D model, for example. This is accessed using the standard Jena graph API. Additional

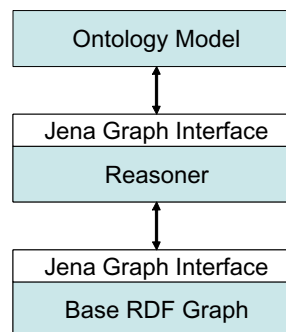


Figure 1. Jena Graph Layering.

components can be layered over that graph with the proviso that each layer exports that same standard graph API. The figure shows an important case of layering where the layered component is a *reasoner* that supports inferences over the underlying graph and makes those inferred edges and nodes visible through its interface. In effect, the reasoner extends

the base graph with any nodes and edges that can be inferred from it. This layering can be continued by adding additional reasoners.

An example is the transitive reasoner. OWL supports transitivity by explicitly attaching a “TransitiveProperty”

annotation to properties. The transitive reasoner adds virtual edges to the base graph based on the existence of transitive markers in that base graph.

Our approach to default values is based on this idea. The key is to treat default values as an inference problem. In particular, suppose there is a query for all triples of the form (A,priority,_). If a search of the base graph returns the empty set, then the default reasoner applies the relevant defaulting rules and returns, for example, the triple (A,priority,0). This assumes that the base graph has been annotated with defaulting markers.

5.3. Methods

Another problem in translating CIM-D to OWL involves methods, which are the signatures for executable procedures associated with a class. An ontology language like OWL has no built-in concept of procedure, so we were again forced to improvise.

There are a number of approaches for modeling methods, but many of them fail in the face of overloaded names and signatures. Our approach requires several ontological elements. There is no room for details here, but, briefly, we define a class for the method, and that class is used as the value for a property of the class containing the method. The arguments of the method become attributes of the method class.

The remaining issue is method inheritance. This is technically an execution time issue, which we consider to be out of scope for this current translation effort.

6. Related Work

The RDF primer [3] makes reference to a project to convert the DMTF CIM model to RDF, however, no reference is given, and a search of the web found no references to this project as of the time of writing of this paper. When it becomes available, a comparison with the work described here will undoubtedly be instructive.

Others have recognized the potential for using an ontology language to represent software engineering information [4]. However, these efforts appear to be in the early stages of development, and the DMTF CIM model does not appear to be one of their targets.

7. Discussion and Conclusions

Translating the DMTF CIM(-D) model into OWL turned out to be more difficult than originally anticipated. In retrospect, using OWL as the common model for our project may not have been a good choice.

OWL's limited notion of scope was a significant problem. The use of a flat, partitioned namespace makes it hard to easily represent a system like CIM where nested naming and implicit reference are inherent in its semantics. Scoping in CIM has associated semantics (basically inheritance), and that is not easily represented in OWL. The only solution may be to flatten the CIM namespace and then find an approach for handling the implicit semantics.

The lack of sophisticated containers in OWL is another major issue. In mapping CIM vectors to lists, for example, the random access nature of vectors is lost. For our purposes, this is not essential, but it is irritating and it may be important in other contexts.

Defaulting is a major problem but we are satisfied with our solution.

In summary, our experience in attempting a natural translation from CIM to OWL was disappointing. We recognize that OWL may not have been designed for this purpose, but that limits its utility outside of the semantic-web context.

Acknowledgements. This material is based in part upon work sponsored by the DARPA and AFRL Rome Labs, under Contract Number F30602-00-2-0608. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

References

- [1] S. Bechjofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Pagel-Schneider, L.A. Stein. "OWL Web Ontology Language Reference W3C Recommendation 10", W3C Working Group, February 2004. <http://www.w3.org/TR/2004/REC-OWL-ref-20040210/>.
- [2] Distributed Management Task Force, Inc., "Common Information Model (CIM) Specification Version 2.2", January 14, 1999. <http://www.dmtf.org/standards/documents/DMI/DSP0005.pdf>
- [3] F. Manola, and E. Miller, "RDF Primer", W3C Working Group, Sept. 5, 2003. <http://www.w3.org/TR/2003/WD-rdf-primer-20030905/>.
- [4] J. E. López de Vergara, V. A. Villagrà, J. I. Asensio, J. Berrocal, "Ontologies: Giving Semantics to Network Management Models," IEEE Network, special issue on Network Management 17(3) (May/June) 2003.
- [5] B. McBride, "Jena: Implementing the RDF Model and Syntax Specification", Proc. of the 2nd Int'l Workshop on the Semantic Web, Hongkong, China, May 1, 2001.

Experiences in Using a Method for Building Domain Ontologies

Ricardo de Almeida Falbo

*Computer Science Department, Federal University of Espírito Santo
Fernando Ferrari Avenue, CEP 29060-900, Vitória - ES - Brazil
falbo@inf.ufes.br*

Abstract. Since 1997 we are working in building domain ontologies. During this period of time, we have developed several ontologies using a systematic approach for building ontologies, first published in 1998, and now called SABiO. In this paper we discuss strong points and weakness of this method for building ontologies, presenting some lessons learned and improvement opportunities.

1. Introduction

Building domain ontologies is not a simple task. Like any complex software modeling activity, to build quality ontologies we need methods and tools to support their development. In 1997, we defined a systematic approach for building ontologies (SABiO), first published in 1998 [1]. SABiO was proposed based on Uschold and King skeletal methodology [2], adding some features to improve it, such as a graphical languages for expressing ontologies, an axiom classification, and the use of competency questions, as proposed by Gruninger and Fox [6]. Since then, we have been using this approach to build several domain ontologies, such as an ontology of software process [1], an ontology of software metrics [3], an ontology of the port domain [4], and an ontology of steel metallurgy, among others.

In this paper we discuss our experience in building domain ontologies using SABiO, focusing on lessons learned and improvement opportunities. Section 2 briefly presents SABiO, and some improvements made along these years of use. Section 3 discusses its strengths, weaknesses and some lessons learned. Section 4 presents some improvement opportunities to evolve SABiO. Finally, section 5 reports our conclusion.

2. A Systematic Approach for Building Domain Ontologies

According to Guarino [5], an ontology is an engineering artifact, constituted by a vocabulary used to describe a certain reality, plus a set of explicit assumptions (formal axioms) regarding the intended meaning of the

vocabulary words. This set of assumptions has usually the form of a first-order logical theory, where vocabulary words appear as unary or binary predicate names, respectively called concepts and relations.

Like any other conceptual modeling activity, ontology construction must be supported by software engineering practices. Thus, we need methods and tools to support ontology engineering. In 1997, we proposed SABiO, a Systematic Approach for Building Ontologies [1], that encompasses the following activities:

- Purpose identification and requirement specification: concerns to clearly identify the ontology purpose and its intended uses, i.e. the competence of the ontology. To do that, competency questions are used.
- Ontology capture: the goal is to capture the domain conceptualization based on the ontology competence. Relevant concepts and relations should be identified and organized. A model using a graphical language and a dictionary of terms should be used to aid communication with domain experts.
- Ontology formalization: aims to explicitly represent the conceptualization captured in a formal language.
- Integration of existing ontologies: during ontology capture or formalization, it could be necessary to integrate the current ontology with existing ones, in order to use previously established conceptualizations.
- Ontology evaluation: the ontology must be evaluated to check whether it satisfies the specification requirements. It should be evaluated in relation to the ontology competence and some design quality criteria, such those proposed by Gruber [7].
- Documentation: all the ontology development must be documented.

During ontology capture, the use of a graphical representation is essential in order to facilitate the communication between ontology engineers and experts. Such representation is basically a language representing a meta-ontology, and thus this language must own basic primitives to represent a domain conceptualization [1].

SABiO proposed the use of LINGO [1], a graphical language for expressing ontologies. In its first version, LINGO had notations for representing concepts, relations,

and properties, and some types of relations that have a strong semantics, such as subsumption and whole-part relations. For each one of these types of relations, a specialized notation was proposed. In fact, this was the striking feature of LINGO and what made it different from other graphical representations: any notation, beyond the basic notations for concepts, relations and properties, aims to incorporate an axiomatization. During its use, some new notations were incorporated to LINGO to address other types of relations, always defining explicitly the axiomatization imposed by them.

More recently, we decided to allow ontology capturing in UML too [4], since UML has also been used as an ontology modelling language [8], and we cannot ignore that UML is a standard and its use is widely diffused. Based on that, we defined a subset of UML's elements that plays the same role of LINGO's notation, i.e., these UML's model elements are applied using the same semantics imposed by the corresponding elements in LINGO. For instance, the epistemological axioms imposed by the whole-part relation are assumed to be incorporated to the ontology when the aggregation notation of UML is used. A lightweight extension of UML was proposed, using stereotypes [4].

A graphical model is useful, but it is not enough to completely capture an ontology. Axioms should be provided in order to fix the semantics of the terms, and to establish domain constraints. To guide axiom definition, SABiO uses an axiom classification that considers two classes of axioms: *derivation axioms*, which allow new information to be derived from the previously existing knowledge, and *consolidation axioms* that define constraints for establishing a relation or for defining an object as an instance of a concept.

Derivation axioms can concern the meaning of the concepts and relations in the ontology, or the way these concepts and relations are structured. When axioms are defined to show constraints imposed by the way concepts are structured, we call them *epistemological axioms*. When they describe domain signification constraints, we call them *ontological axioms*. This distinction is important to guide the ontology engineering defining axioms. Epistemological axioms can be assumed to be captured by the graphical notation, and should not be explicitly written. Ontological axioms, in turn, are not captured by the graphical notation, and need to be explicitly defined. In Figure 1, we show part of the software process ontology defined in [1], written in UML. In this figure, the aggregation notation imposes some axioms, such as:

$$\begin{aligned} \forall a \neg \text{subActivity}(a,a) \\ \forall a1,a2 \text{ subActivity}(a1,a2) \rightarrow \neg \text{subActivity}(a2,a1) \\ \forall a1,a2,a3 \text{ subActivity}(a1,a2) \wedge \text{subActivity}(a2,a3) \rightarrow \\ \text{subActivity}(a1,a3) \end{aligned}$$

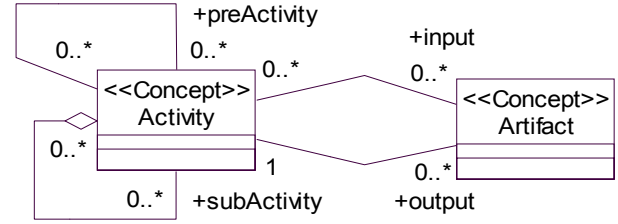


Figure 1 - Part of the software process ontology.

These axioms are part of the mereological theory, which says that whole-part relations are irreflexive, anti-symmetric and transitive, respectively, and do not need to be written by the ontology engineer, since they are epistemological axioms.

In the same ontology, however, there is the following ontological axiom:

$$\forall a1,a2,s \text{ input}(s,a2) \wedge \text{output}(s,a1) \rightarrow \text{preActivity}(a1,a2)$$

This axiom does not refer to the way concepts are structured, and thus, cannot be captured by the graphical notation. It is an ontological axiom and must be written down by the ontological engineer. This way, the distinction between epistemological and ontological axioms indicates which axioms must be written by the ontology engineer.

Going back to the activities of the ontology development process shown in Figure 1, to formalize ontologies, SABiO suggests the use of first order logics, and gives some guidelines to perform this step [1].

In ontology evaluation, SABiO suggests checking the ontology against its competency questions, and to verify some quality criteria, as pointed early.

Finally, for documentation purposes, SABiO advocates the use of hypertexts. Using a hypertext, concepts can be easily linked to relations, properties, ontology diagrams, dictionaries of terms, axioms, and competency questions. This way, people can browse the ontology to learn about the domain.

3. Strong Points, Weakness and Lessons Learned

After we had used SABiO in several ontology developments, we can point out some benefits and some weakness of the method.

Concerning strong points of SABiO, we can highlight:

- The set of activities, artifacts and guidelines proposed by the ontology development process of SABiO showed to be good. It can be considered part of a standard software process for building ontologies, but we need more.
- The use of competency questions showed to be very useful to guide ontology capturing, formalization and evaluation. Concepts, relations, properties and axioms in an ontology should be those necessary and

sufficient to address the competency questions, as pointed by Gruninger and Fox [6].

- The use of a graphical language for expressing ontologies proved to be essential for ontology capture. It is very hard to communicate with domain experts without it. More over, the epistemological axioms incorporated to the graphical notation free ontology engineers to concentrate in some classes of axioms, in spite of having to consider all of them.
- The axiom classification also proved to be a good guideline to drive the axiom definition. Based on it, ontology engineers can inspect the world looking for axioms that consider the structuring of the concepts and relations (the epistemological axioms), their meanings and constraints (the ontological axioms) and the integrity laws that govern them (the consolidation axioms). But the first class of axioms do not need to be written down.
- Hypertext proved to be an adequate format for documenting ontologies. Using hypertexts, ontologies can be easily browsed, and people can use them to learn about the domain.

But SABiO has also weaknesses, such as:

- SABiO does not address important activities of a software process, as recommended in Software Engineering, such as planning and configuration management. Regarding the last, in fact, SABiO says nothing about ontology evolution.
- Concerning competency questions, SABiO says nothing about formal competency questions. We think they are very important. But we need tools for verifying ontologies in the light of them.
- Although LINGO has a strong semantics, it is “another modeling language”. This is a recurrent claim. Many ontology engineers do not know it, and sometimes use it in an inappropriate way. Several times, we notice that notations were not being correctly used, and the models were not well interpreted by ontology engineers. When we started to use UML as modeling language, some of these problems attenuated. On the other hand, sometimes, ontology engineers with background in software engineering used some UML constructions that are not expected in ontology building, and consequently without precise semantics.
- As to axiom classification, sometimes ontology engineers have doubts about how to classify an axiom. The most common problems are about some epistemological axioms, like those imposed by cardinalities. Cardinalities, for instance, express domain constraints, and thus ontology engineers tend to classify them as ontological axioms, in spite of they are related to structural concerns.
- A first order predicate logic language for formalizing ontologies is good due to its expression power. But it

is difficult to evaluate an ontology formalized using it, since we do not have inference engines capable to do that. Other languages, such DAML+OIL [9] and KIF [10], could be better choices, since we can use an inference engine to verify the ontology. Competency questions could be formalized and submitted to the inference engine to check if the ontology satisfies them. But some of them, like DAML+OIL, are less expressive languages.

- Ontology integration in SABiO is extremely superficial. Nothing is said about consistency and coherence of the model elements imported to a new ontology.

Finally, we can enumerate some lessons learned. First, like any other software product, ontology building must be conducted as a quality software process. As a software process, we need tools to support ontology building. Ideally, such tools have to allow competency question definition and formalization, ontology capture using a graphical language, axiom definition and formalization, ontology integration, ontology verification and validation, ontology documentation, and ontology evolution.

Second, especially in ontology capturing we need to achieve consensus from experts. Books, papers, manuals, web pages and other literature sources are very important for capturing an ontology, but they are not enough. We need experts, and need to achieve consensus between their positions. In this process, Gruber’s minimum ontological commitment criterion [7] is very useful. In all work we have been done, we needed to apply this criterion in order to achieve consensus.

Third, in ontology building, evaluation regards the set of activities that ensure that the ontology concepts, relations, properties and axioms answer appropriately the competency questions. Two questions have to be answer: “Are we building the ontology right?” and “Are we building the right ontology?” The first one regards ontology verification, the second ontology validation. In both cases, evaluation implies to check each competency question, looking if it is being correctly answered. For this purpose, we need tools to support those activities, since they are hard tasks to be done manually. Particularly in ontology validation, experts are essential. In ontology validation not only we are checking if the competency questions are being correctly answered, but we are also checking if the competency questions actually pose the right questions for the ontology purpose.

Finally, although hypertexts proved to be an excellent way to document ontologies, we need tools to automate, at least partially, their construction. Ontology engineers spend a substantial amount of time developing the ontology documentation. Documentation functionalities integrated into an ontology editor is an important opportunity to improve productivity.

4. Improvement Opportunities

Based on the weaknesses of SABiO, we can devise some improvements to evolve it to a better approach for building ontologies:

- It is worthwhile to define a standard software process for building ontologies, in the sense of Software Engineering. Planning activities and methods to do that should be investigated. There are few works addressing this important issue. Metrics for evaluating ontology development should also be provided. Software Engineering experience can serve as basis, but we need to adapt it to better fit ontology development.
- Regarding a modeling language for expressing ontology, we think that the use of a lightweight extension of UML, such that proposed in [11] is a promising way. We are now studying how to incorporate it to SABiO.
- We should refine the guidelines for classifying axioms in order to clarify the categories. Also, we are studying how relation meta-properties, such as transitivity and symmetry, can be integrated to our axiom classification. These are very frequent axioms, and so it is worthwhile to better support their capture.
- During ontology formalization, competency questions should be formalized. In ontology evaluation, they should be submitted to an inference engine to check if the ontology satisfies them.
- SABiO needs to better address ontology integration. In its current version, all important decisions are left to the ontology engineers. We need to better study this activity to improve the guidelines offered to it.
- SABiO does not consider ontology maintenance or evolution. Since we are now working in some ontology evolutions (this is the case of the software process ontology [1]), we intend to improve SABiO with practical guidelines to address ontology evolution.

5. Conclusions

Building domain ontologies is not a simple task. We need methods, tools and guidelines to drive ontology engineers in performing their activities. Software engineering practices should be incorporated to ontology development, and SABiO goes a step ahead towards a defined ontology development process.

In this paper we presented some reflections regarding the strengths and weaknesses of SABiO, and discussed some lessons learned and improvement opportunities.

Our experience in ontology development highlights an important issue concerning tool support. We would not be

able to scale up ontology building without good ontology editors. Fortunately, now there are some of them available, such as OIEd [12]. We are also working on ODEd [4], an ontology editor that minimizes some of the reported problems, such as formalization and evaluation.

Acknowledgments

This work was accomplished with the support of CNPq, an entity of the Brazilian Government reverted to scientific and technological development.

References

- [1] R.A. Falbo, C.S. Menezes, A.R.C. Rocha. "A Systematic Approach for Building Ontologies". Proc. of the 6th Ibero-American Conference on Artificial Intelligence, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.
- [2] M. Uschold, M. King. "Towards a Methodology for Building Ontologies", Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI'1995.
- [3] R.A. Falbo, G. Guizzardi, K.C. Duarte. "An Ontological Approach to Domain Engineering". Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, SEKE'2002, pp. 351- 358, Ischia, Italy, 2002.
- [4] P.G. Mian, R.A. Falbo. "Building Ontologies in a Domain Oriented Software Development Environment". *Proceedings of the IX Argentine Congress on Computer Science*, pp. 930 – 941, La Plata, Argentina, 2003.
- [5] N. Guarino. Formal Ontology and Information Systems. In N. Guarino (Ed.), *Formal Ontologies in Information Systems*, IOS Press, 1998.
- [6] M. Grüninger, M.S., Fox. Methodology for the Design and Evaluation of Ontologies. *Technical Report*, University of Toronto, 1995.
- [7] T.R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. Journal Human-Computer Studies*, 43(5/6), p. 907-928, 1995.
- [8] S. Craneffeld, M. Purvis. UML as an Ontology Modelling Language, In *Proceedings of the IJCAI-99, Workshop on Intelligent Information, 16th International Joint Conference on AI*, Stockholm, Sweden, July 1999.
- [9] D. Connolly, F. van Harmelen, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein. "DAML+OIL Reference Description", December 2001.
- [10] M.E. Genesreth, R.E. Fikes. "Knowledge Interchange Format, Version 3.0 Reference Manual". Tech. Rep. Logic-921, Computer Science Dept., Stanford University, 1992.
- [11] G. Guizzardi, H. Herre, G. Wagner, "Towards Ontological Foundations for UML Conceptual Models", 1st International Conference on Ontologies, Databases and Application of Semantics (ODBASE'2002), Irvine, California, USA, 2002.
- [12] S. Bechhofer, I. Horrocks, C. Goble, R. Stevens. "OIEd: a Reason-able Ontology Editor for the Semantic Web". Working Notes of the 14th International Workshop on Description Logics (DL-2001), pp.1-9, USA, August 2001.

Learning Materials Ontology and Semantic Web: a case study in Educational Domain

Moysés de Araújo

Maria A. G. V. Ferreira

Escola Politécnica – Universidade de São Paulo

e-mail: [moyses.araujo,maria.alice.ferreira]@poli.usp.br

Abstract. In order to restructure the World Wide Web there is a new technology, known as Semantic Web, being developed. It aims to structure and organize information for more intelligent and effective search, making use of the ontology concept. This work presents an ontological modeling for learning materials case study, based on Semantic Web technology for an educational platform named CoL (Courses on LARC). This proposal extends such platform, adding to it the possibility of organizing and structuring its learning materials, making possible more “intelligent” and structured searches on the materials as well as making possible the reuse of the materials contents.

1. Introduction

The Web is becoming the world virtual library, where information on any subject is available at any time and anywhere, with or without cost, creating chances in some areas of the human knowledge, amongst which the Education.

However, with the revolution that the Web has made possible in the access to the information, new boarding can be made to improve the quality and to develop the efficiency of the education based on the Web. Among them can be cited [1]:

- Sharing and reuse of learning materials between applications;
- Learning materials structuring through common reference points;
- Computers qualification so that they can understand and interpret the learning materials.

Currently, no automatic form exist to share and to reuse learning material between applications. Most of the systems uses different formats, languages and vocabularies to represent and to store these materials. For this reason, teachers have a great problem to address:

- How to find information on learning materials to illustrate their lessons, destined to an auditorium

ever more demanding and used to television and Internet?

- How to reuse the existing material easily, without having to produce new material, each time?

A solution to this problem is to develop educational applications in which the learning material is based on ontologies.

In this work, a method is discussed that allowed to extend a conventional educational platform (CoL) [2] to search and retrieve learning material, through ontologies. The work is structured as follows. In item 2, some concepts about Semantic Web and ontologies are provided. In items 3 and 4, the CoL platform and the learning material ontology are, respectively, introduced and in item 5, the conclusion of the work is presented.

2. Education based on Semantic Web and Ontologies

“The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation”. In these words, Berners-Lee et al. define the Semantic Web [3].

Ontologies is an essential technology for the Semantic Web. In Gruber [4], Studer et al. [5], Swartout [6] and Chandrasekaram [7] can be found other definitions and discussions on ontologies. In the scope of this work, it can be said that ontology is a set of concepts, terms and relations that can be used to describe some area of knowledge or to construct its representation. Through these terms, facts can be described on a certain domain, so that they can be understood by the machines, making intelligent search in the Web possible.

To structure the learning materials with common reference points, concepts and relations must be based on a standard vocabulary. With this vocabulary, and using the ontologies, all the parts composing the learning materials can be kept linked together.

So, if computers have to understand and interpret the learning materials, the pages that compose the

applications need to contain semantic tags, established in the terms defined for one or more ontologies. These annotations make possible for structured search to be carried through learning materials and objects.

In this context, the following definition for learning object is adopted: "Learning object is any digital resource that can be reused to support learning" [8]. So, learning objects are elements of a new type of computer-based instruction grounded in the object-oriented paradigm of computer science. Object-orientation highly values the creation of components (called "objects") that can be reused in multiple contexts [9]. This is the fundamental idea behind learning objects: instructional designers can build small (relative to the size of an entire course) instructional components that can be reused a number of times in different learning contexts.

This idea makes possible for the learning materials to be centralized in the Web in many different formats, such as hypertext, video, animations, simulations etc. Thus, the Web-based learning systems will have to adopt a new approach in their development, the use of the technologies that form the base of the Semantic Web (XML, RDF and ontologies), with the use of learning objects.

However, to attend these requirements languages that represent semantics of the information on the Web are necessary to enable the data exchange between heterogeneous environments. Several languages for ontologies were developed in the context of the Semantic Web, among them the language DAML+OIL, a new proposal of the consortium W3C, to serve as starting point for the activities of the Semantic Web. According to Horrocks [10] "DAML+OIL is an ontology language, and as such is designed to describe the structure of a domain. DAML+OIL takes an object oriented approach, because the structure of a domain can be described in terms of its classes and properties".

3. The CoL Platform

The CoL (Courses on LARC) platform is a Distance Learning system, developed by LARC Laboratory of the Polytechnic School of the University of São Paulo.

A course in CoL is formed by modules, disciplines and groups. A group can have several disciplines and a discipline can have several modules. Module is the basic unit of a course in CoL. It is considered an abstraction of a concept, a chapter of a book or any subject. It should be formed by one or more HTML pages linked to each other and can contain any kind of file related with the content, like video, sound, images, animations, etc.

In order to improve the understanding of the CoL structure, an example is provided as follows. Suppose a post graduation course on Semantic Web, formed by any group sample of students. The disciplines to be attended can be: XML – Basic Concepts, RDF/RDF Schema,

Ontologies and Languages for Semantic Web. The discipline XML – Basic Concepts can have the structure shown in Fig. 1. The same structure can be applied to the other disciplines. The numbered items represent the programmatic content of each module. Each item is an HTML page, as shown by the links in Fig. 1, developed by the teacher.

Module 1- Markup Languages

- 1.1 – [Definition](#)
- 1.2 – [SGML](#)
- 1.3 – [HTML – Definition](#)
- 1.4 – [HTML – Format](#)
- 1.5 – [Summary](#)
- 1.6 – [Tests](#)

Fig. 1 – A module of discipline XML – Basic Concepts

4. Building a Learning Material Ontology

The main goal in constructing an ontology for educational systems is to make possible a representation of the semantics of the educational materials that are stored in the educational platform, so that they can be reused, shared, structured and so, that the users of this platform (teachers, learners, administrators) can perform queries wisely. To extend CoL platform, the concept of its educational material need to change to introduce semantics. The ontology will provide a vocabulary so that the material can be annotated, as well as allowing for a set of relationships to be established between the terms of the vocabulary, to provide inferences in the knowledge base.

For the introduction of a learning material ontology following steps must be taken:

Step 1 - To establish competency questions for learning materials - the ontology must answer, for example, competency questions like:

1. Which learning materials compose the platform?
2. What are the requirements of some learning material?
3. Are there similar learning material in the platform?
4. What are the types of learning objects that compose the learning materials?

To answer the competency question 1 it is necessary to incorporate new concepts concerning learning objects and learning materials. These elements will relate with the modules. Thus, it can be said that modules are formed by learning materials that, in turn, are constituted by learning objects. Fig. 2 presents the new elements. The following relationships exist between the concepts of module and learning material:

- the relationship `hasMat` denotes that a module is formed by learning materials. This relation has the cardinality (1,n) that determines the axiom: “Every module is formed by one, or more, learning materials”.
- the relationship `isInMod` indicates that the learning materials compose modules.

In Fig. 1, Module 1 will be composed, now, by six learning materials (numbered items 1.1 to 1.6).

Competency questions 2 and 3 are answered by the following relationships between learning materials:

- The relationships `isRequisiteOf` and `hasRequisite` are inverse relations, and say that if a learning material B has a requisite A, then A is requisite of B.
- The same reasoning is applied for the relationships `isSimilarTo` and `isSimilarOf`. A learning material is similar to another one, when the same subject, for example, equations visualization, can be treated through a text, a graph or an animation.

Competency question 4 mentions the learning objects that form the learning materials. According to the Learning Technology Standards Committee of the IEEE specification [11] and the Global IMS Learning Consortium specification [12], the types of learning objects are: Exercises, Simulation, Questionnaire, Diagram, Figure, Graph, Index, Table, Narrative Text, Examination, Experience, Problem Enunciation and Auto Evaluation. These concepts are represented by the following relations:

- the relationship `hasObject` denotes that the learning materials have one or more learning objects. This relationship has cardinality (1,n) and determines the following axiom: “Every learning material has one or more learning objects”.
- the relationship `isInMat` is the inverse relationship of `hasObject` and indicates that the learning objects can belong to the learning materials. The relationships of type `subclassOf` that appear in Fig. 2 indicate that Exercise, Simulation, Problem, Text and others not shown in Fig. 2 are Learning Object specializations.

Step 2 – Specification of the ontological knowledge base in DAML+OIL - In DAML+OIL the objects are described through classes (named `daml:Class`). To relate individuals with each other, properties (named `daml:ObjectProperty`) are used, and class attributes are described through properties of the data (named `daml:DatatypeProperty`). The property’s range and domain are stated by `rdfs:range` and `rdfs:domain`, respectively. For example, DAML+OIL codification of the concept Learning Material, showed in Fig. 2, is presented in Fig. 3. Fig. 4 presents

the instance of Module concept, showed in Fig. 1, in DAML+OIL.

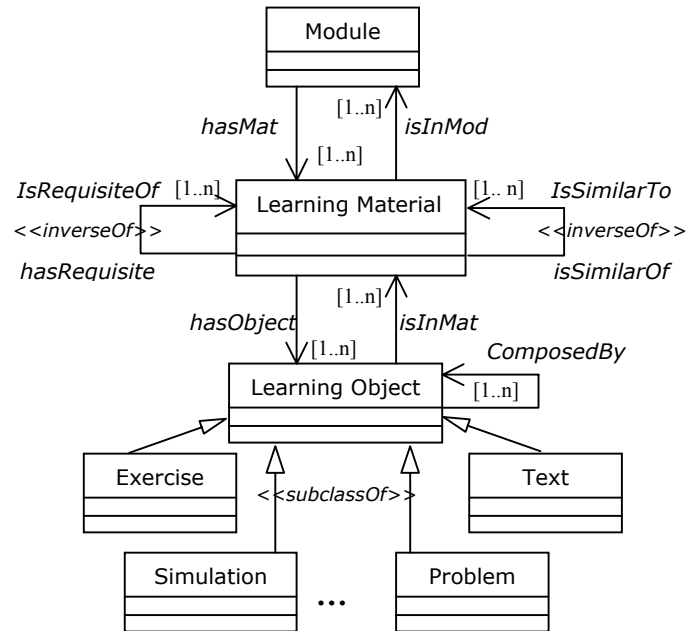


Fig. 2 – Learning Materials Ontology

```
<daml:Class rdf:ID="Learning_Material">
</daml:Class>
<daml:DatatypeProperty rdf:ID="code">
<daml:domain rdf:resource="#Learning_Material"/>
<daml:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</daml:DatatypeProperty>
<daml:DatatypeProperty rdf:ID="name">
<daml:domain rdf:resource="#Learning_Material"/>
<daml:range
rdf:resource="http://www.w3.org/2000/10/XMLSchema#string"/>
</daml:DatatypeProperty>
<daml:ObjectProperty rdf:ID="isInMod">
<rdfs:domain rdf:resource="#Learning_Material"/>
<rdfs:range rdf:resource="#Module"/>
<daml:minCardinality>1</daml:minCardinality>
<daml:maxCardinality>n</daml:maxCardinality>
</daml:ObjectProperty>
<daml:ObjectProperty rdf:ID="hasObject">
<daml:domain rdf:resource="#Learning_Material"/>
<daml:range rdf:resource="#Learning_Object"/>
<daml:minCardinality>1</daml:minCardinality>
<daml:maxCardinality>n</daml:maxCardinality>
</daml:ObjectProperty>
```

Fig. 3 – Partial Code for Learning Material in DAML+OIL

One should notice that the ontology forms an ontological knowledge base, that can have its information in one or more files: one file stores elements like vocabulary, relationships and attributes and the other file stores the

generated instances. This separation aims to facilitate ontology maintenance; therefore, the instances archive is constantly modified.

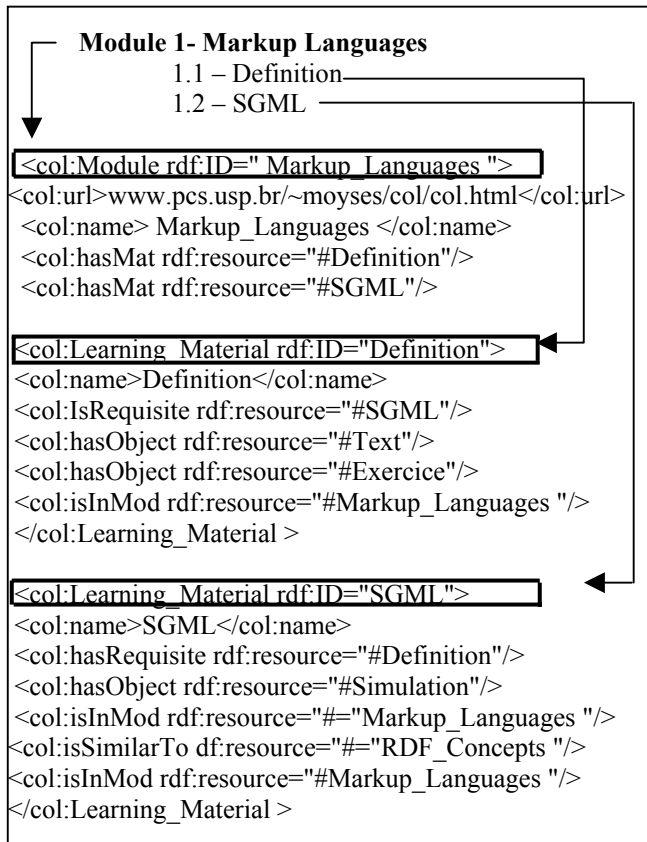


Fig.4 – Learning Material instances

Step 3 - To develop the search engine for the ontological knowledge base - To carry through the search in the ontological knowledge base a search engine is needed. This engine will verify the relationships and ontological instances codified in the ontological language. Since DAML+OIL is chosen as the representation language in Step 2, one should be able to use any DAML+OIL reasoner. In this work, a prototype engine named AQ Search was used. AQ Search is available in the official page of the DAML (DARPA Agent Markup Language - <http://www.daml.org>). This search engine, developed with Java tools, is composed by a graphical interface, that allows the users to carry through the research and to return the results, and by an agent to process the research in the ontological knowledge base.

5. Conclusion

The current work of the Semantic Web community is directed mainly for the representation of information in the World Wide Web, so that these pieces of information can be used by the machines, not only to show

information but also for tasks automation, more intelligent integration, sharing, research and reuse of information between the applications. In the educational scope, Distance Education is an important goal to pursue and the Semantic Web offers optimistic perspectives. One system prototype was modeled according to the methodology described in Araújo [13].

References

- [1] V. Devedzic. "What does current web-based education lack.". Proceedings of the IASTED International Conference APPLIED INFORMATICS. Innsbruck, Austria, Feb 2002.
- [2] R. M. Silveira et al. COL – Ferramenta de Apoio ao Ensino. Technical Report. LARC Laboratory. University of São Paulo. São Paulo, 2002.
- [3] T. Berners-Lee.; J. Hendler; O. Lassila. "The Semantic Web". Scientific American. v. 284, n. 5, 2001, pp. 28-37.
- [4] T. R. Gruber. "Toward Principles for the Design of Ontologies Used for Knowledge Sharing". In: Nicola Guarino and Roberto Poli (Ed.). Formal Ontology in Conceptual Analysis and Knowledge Representation.. Kluwer Academic Publishers, 1993.
- [5] R. Studer; V. Benjamins; D. Fensel. "Knowledge Engineering: Principles and Methods". IEEE Transactions on Data and Knowledge Engineering, v.25, n.1-2, 1998, pp.161-197.
- [6] W. Swartout. Ontologies. IEEE Intelligent Systems. v. 14, n. 1, Jan. 1999, pp. 18-19.
- [7] B. Chandrasekaran; R. Josephson; V. R. Benjamins. "What Are Ontologies, and Why Do We Need Them?". IEEE Intelligent Systems. v. 14, n. 1, Jan. 1999, pp. 20-25.
- [8] D. A. Wiley. "Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy". In D. A. Wiley (Ed.) The instructional use of learning objects. 2001. <http://reusability.org/read/chapters/wiley.doc>. (link 25-03-2003).
- [9] O.-J. Dahl, K. Nygaard, "Simula: an Algol-based simulation language, Comm. of the ACM, v.9, n.9, Sept. 1966, pp. 671-678.
- [10] I. Horrocks. "DAML+OIL: a Reason-able Web Ontology Language". In Proc. of EDBT2002, in Lectures Notes in Computer Science, No. 2287, March 2002, pp. 2-13.
- [11] Learning Technology Standards Committee of the IEEE, New York, 15/07/2002. Draft Standard for Learning Object Metadata. <http://ltsc.ieee.org/wg12/index.html>. (link 9-04-2003).
- [12] IMS Global Learning Consortium, "IMS Learning Resource Meta-Data XML Binding", Version 1.2.1 Final Specification, 28 Set. 2001. <http://www.imsglobal.org>. (link 20-02-2003).
- [13] M. Araújo. Educação a Distância e a Web Semântica: Modelagem Ontológica de Materiais e Objetos de Aprendizagem para a Plataforma COL. PHD Thesis, . University of São Paulo, São Paulo, 2003.

Non-taxonomic Relations in Semantic Service Discovery and Composition

Michael Lutz

*Institute for Geoinformatics, Robert-Koch-Str. 26-28, D-48149 Münster, Germany
m.lutz@uni-muenster.de*

Abstract. Existing approaches for semantic web service discovery are often based on subsumption reasoning, i.e. on evaluating taxonomic relations between the ontology concepts used in the service descriptions. If these concepts are not related taxonomically, no match is found. In this paper we present an approach that recognizes the importance of non-taxonomic relations. To increase recall, it uses operations defined in the domain ontology as a common template for semantic descriptions of web service inputs and outputs. The problems of existing approaches and the benefits of the proposed solution are illustrated using a real-world example and a state-of-the-art tool for semantic service discovery.

1. Introduction

The ability of composing several simple web services into more complex ones is often seen a main advantage of service-oriented computing. An important task during service composition is to discover services with appropriate functionality whose inputs match – syntactically *and* semantically – the outputs¹ of adjacent services.

With a growing number of web services available, the task of service discovery is becoming increasingly important. Searches in current service registries (e.g. UDDI) are based on keywords, fixed taxonomies and syntactic service descriptions like WSDL. As this approach leads to numerous problems causing low precision and recall [1] approaches based on reasoning with semantic service descriptions that refer to ontologies have been proposed (e.g. [2-4]). Most of the semantic descriptions proposed are closely linked to or inspired by OWL-S [5].

Many of these approaches are based on evaluating subsumption relationships between the ontology concepts the service descriptions refer to. This means that in cases where ontology concepts are related via *non-taxonomic* relations rather than simple taxonomies service discovery based on subsumption reasoning will not always yield the desired results. This will be illustrated using the approach for service discovery described in [2].

In this paper, we present a procedure for deriving alternative semantic descriptions that take into account the importance of non-taxonomic relations. The goal is to improve recall during service discovery based on subsumption reasoning.

The remainder of the paper is structured as follows. We introduce a motivating example in section 2 and provide relevant background for our work in section 3. In section 4 the investigated approach to semantic service discovery is introduced and the problems connected with subsumption reasoning are discussed. Our proposed solution is introduced in section 5 and discussed in section 6.

2. Motivating Example

To illustrate current problems during ontology-based service discovery and our proposed solution, we use the following example throughout the paper. Susan is a service provider who wants to build a complex service that computes the distance between two industrial plants. She has already found a service² providing the location of a given industrial plant as a point geometry. The point is represented as a complex type consisting of two geographic coordinates (latitude, longitude).

type point (latitude : double, longitude : double)

Susan uses this output type as a requirement during her search for services that compute distances between points. For our example, we consider two candidate services³ each of which provides an operation for computing the great circle distance (gcd) between two points (each represented by two coordinates):

gcd1 (lat1: double, long1: double,
lat2: double, long2: double) : double

gcd2 (x1: double, y1: double,
x2: double, y2: double) : double

The input parameters of gcd1 can be any geographic coordinates; those of gcd2 represent coordinates in a spe-

¹ Or vice versa, depending in which order the composite service is composed.

² adapted from <http://acegis.e-blana.com/PreEmergencyPlanService/PreEmergencyPlanService.asmx>

³ adapted from <http://samples.bowstreet.com/bowstreet5/webengine/xmethods/gcd/Action!getWSDL>

cific projected coordinate reference system (“WGS84 / UTM zone 11N” in our example). Both operations return the (great circle) distance in miles.

If Susan relied only on syntactic descriptions of the operations (i.e. their signatures) in her search, both `gcd1` and `gcd2` would seem appropriate, because the result of the “plant location” service (two doubles) could be fed into both services. However, the geographic coordinates representing the plant’s location would not be interpreted correctly in `gcd2`, leading to wrong results. Methods and tools for semantic service discovery have been introduced to avoid such mistakes.

3. Background

In this chapter, we introduce the logic notation used in this paper as well as the domain and application ontologies employed for our example.

The Notation. We use a Description Logics (DL) notation to define concepts. DL is a family of knowledge representation languages that are subsets of first-order logic. They provide the basis for the Ontology Web Language (OWL), the proposed standard language for the Semantic Web [6].

Of the available DL language features, we use *concept definition* ($C \equiv D$) and *concept inclusion* ($C \sqsubseteq D$) in this paper, and the following constructors for concepts (C , D) and roles (R , S):

$D \rightarrow C \sqcap D$	(intersection)
$\forall R.C$	(value restriction)
$\exists R.C$	(existential quantification)
$\leq (\geq) n R$	(number restrictions)
$S \rightarrow R^-$	(inverse)

Of the available inference procedures, the possibility to compute subsumption relationships is of special importance for our work. For a more detailed introduction to DL languages see [7].

Domain Ontology. Our example is drawn from the domain of geographic information (GI). Therefore, we use a collection of models from this domain as a basis for a domain ontology, the 19100 series of ISO standards. Here, we only consider a small and simplified extract of this model (for the relevant standards see [8, 9]) containing the following concepts:

- Points (`GM_Point`) can (but need not) be associated to one coordinate reference system (`coordRefSys`) of type `SC_CRS`. They have $n (\geq 1)$ coordinates, which are represented as `Numbers`.
- Geographic coordinate reference systems (`Geog_CRS`) are a specific family of coordinate reference systems, one of whose members is denoted as `EPSG4326`.

- `EPSG26911` denotes a coordinate reference system (WGS84 / UTM zone 11N) that is not a member of the family of geographic coordinate reference systems. A schematic illustration of the model is given in Figure 1.

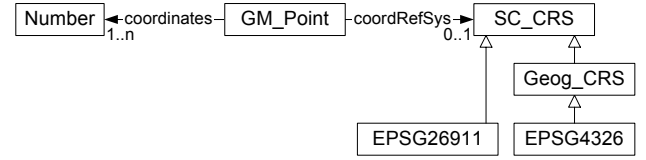


Figure 1: Extract of the domain ontology

In order to use the ISO model as a domain ontology for web service discovery its semi-formal UML models and natural language definitions have to be translated into DL. The extract of the domain ontology is described by the following axioms:

$T \sqsubseteq (\forall \text{coordinates.Number})$
 $\text{GM_Point} \sqsubseteq (\forall \text{coordRefSys.SC_CRS}) \sqcap (\leq 1 \text{ coordRefSys}) \sqcap (\geq 1 \text{ coordinates})$
 $\text{Geog_CRS} \sqsubseteq \text{SC_CRS}$
 $\text{EPSG4326} \sqsubseteq \text{Geog_CRS}$
 $\text{EPSG26911} \sqsubseteq \text{SC_CRS}$

Application Ontology. The application ontology contains more specific concepts, which can be used for annotating syntactic service descriptions and for specifying a requester’s requirements. In our example, the output of the first service discovered by Susan is annotated with the concept `Plant_Location`, the input parameters of `gcd1` and `gcd2` with the concepts `Geog_Coord` and `UTM_Coord`:

- `Plant_Location` is a specific kind of point, which has a specific geographic coordinate reference system (`EPSG4326`) and two coordinates.
- `Geog_Coord` is a `Number` that is a coordinate of a specific kind of point (`GM_Point1`). This point has two coordinates and some geographic coordinate reference system (`Geog_CRS`).
- `UTM_Coord` is a `Number` that is a coordinate of a different kind of point (`GM_Point2`), which also has two coordinates, but `EPSG26911` as a coordinate reference system.

In DL syntax, this is expressed by the following axioms:

$\text{Plant_Location} \sqsubseteq (=1 \text{ coordRefSys}) \sqcap (\forall \text{coordRefSys.EPSG4326}) \sqcap (=2 \text{ coordinates})$
 $\text{GM_Point1} \sqsubseteq (=1 \text{ coordRefSys}) \sqcap (\forall \text{coordRefSys.Geog_CRS}) \sqcap (=2 \text{ coordinates})$
 $\text{isCoordinateOf} \sqsubseteq \text{coordinates}^-$
 $\text{Geog_Coord} \sqsubseteq \text{Number} \sqcap (\exists \text{isCoordinateOf.GM_Point1})$
 $\text{GM_Point2} \sqsubseteq (=1 \text{ coordRefSys}) \sqcap (\forall \text{coordRefSys.EPSG26911}) \sqcap (=2 \text{ coordinates})$
 $\text{UTM_Coord} \sqsubseteq \text{Number} \sqcap (\exists \text{isCoordinateOf.GM_Point2})$

Note that the definitions of *Geog_Coord* and *UTM_Coord* use an inverse relation (*isCoordinateOf*) to express that they are the range of the relation *coordinates* and that the domains of that relation are the previously defined concepts *GM_Point1* and *GM_Point2*, respectively.

The application ontology is illustrated in Figure 2.

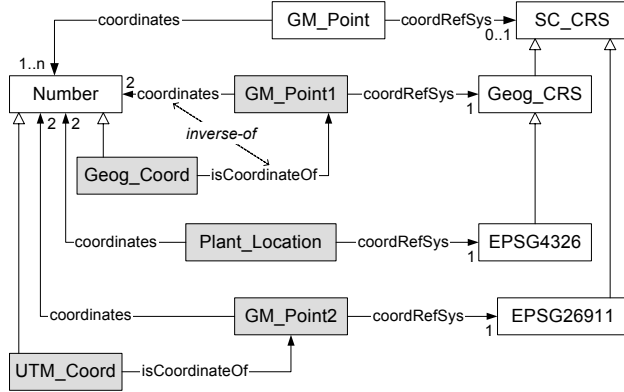


Figure 2: Extract of the application ontology (domain ontology concepts are shown in white)

Even though it is not stated explicitly in the axioms, it can be inferred that *Plant_Location* is subsumed by *GM_Point1*, and that both *GM_Point1* and *GM_Point2* are subsumed by *GM_Point* (Figure 3).

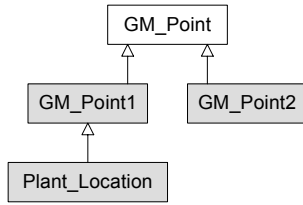


Figure 3: Inferable taxonomic relationships between application and domain concepts

4. Problems During State-of-the-art Semantic Service Discovery

In this section, we introduce a state-of-the-art approach to semantic service discovery based on subsumption reasoning and illustrate some of the problems associated with it.

The Semantic Services Matchmaker. The *Semantic Services Matchmaker (SSM)* [2] is a tool for matching between semantic descriptions of services and user requests in order to enhance the discovery facilities of UDDI. For referring to the ontology classes required for the matchmaking algorithm described below, the SSM uses an extension to WSDL called *Web Service Semantic Profile (WSSP)*, which is inspired by the OWL-S Service Profile [5].

The SSM is based on the LARKS and the OWL-S matching algorithms [4, 10]. Specifically, it adopts the

LARKS approach to offer a series of five filters, which can be combined to implement three different degrees of match. For our work, we are specifically interested in the *I/O Type Filter*. For an in-depth discussion of all filters, see [10].

The *I/O Type Filter* checks whether the definitions of the input and output parameters match. In the semantic service description used by the SSM, parameter types of inputs and outputs are defined as ontology classes. Matching in this filter is mainly based on identifying subsumption relationships between input and output parameters⁴.

Problems. The SSM generates a template from the service’s WSDL document, in which the user has to select a matching ontology concept for each input and output parameter. We assume for our example that the inputs of *gcd1* are annotated with the *Geog_Coord* and the inputs of *gcd2* with the *UTM_Coord* concept from the application ontology.

Susan, on the other hand, uses the *Plant_Location* concept in her search for a point distance service that accepts the output of the “plant location” service. However, as there exist only *non-taxonomic* relations between *Plant_Location* and *Geog_Coord* or *UTM_Coord* (Figure 2), Susan discovers neither *gcd1* nor *gcd2*.

5. Template Operations as a Basis for Semantic Service Descriptions

In order to avoid problems as those described above we introduce an alternative way to semantically annotate web services that is not based directly on syntactic (WSDL) descriptions. Rather, we use a *template operation* as a common basis for specifying service capabilities and requirements.

Template operations are part of the domain ontology. In the geo-spatial domain, they can also be derived from the ISO 19100 series of standards, which also include operations, e.g. a distance operation between points:

distance(*p1* : *GM_Point*, *p2* : *GM_Point*) : Distance

The parameters of the template operation serve as a basis for defining more specific concepts, which can be used for describing inputs and outputs of a specific service. These concepts must always have a taxonomic relation to the domain concepts describing the template operation’s parameters. For example, the inputs to our example operations *gcd1* and *gcd2* can be annotated using the application ontology concepts *GM_Point1* and *GM_Point2*, respectively. Both concepts are subsumed by the domain concept *GM_Point*, which describes the template operation’s input.

⁴ The algorithm also considers the number of parameters. For details see [2].

Likewise, a requester like Susan can use the inputs and outputs of template operations as a basis for describing her requirements. She can use the *Plant_Location* concept as a requirement in her query because it is subsumed by *GM_Point* and thus in accordance with the distance template operation. Because of the inferable taxonomic relations between the used concepts (Figure 3) Susan now correctly discovers *gcd1* but not *gcd2* when using service discovery algorithms based on subsumption reasoning (like that employed in the SSM).

6. Discussion and Future Work

We have introduced a procedure for generating descriptions of the input and output parameters of web services based on *template operations*, which recognizes the importance of non-taxonomic relations for service discovery and composition. When these descriptions are used with existing algorithms based on subsumption reasoning, recall can be increased.

In future work we are aiming to test our procedure with additional real-world examples. Also, several parts of the procedure need to be refined:

- With the presented approach an ontology concept can refer to *several* input parameters rather than just one. This means that semantic service descriptions, too, should allow the annotation of several parameters with *one* ontology concept. The WSSP templates suggested in [2] seem presently not to provide this option.
- We currently assume that an appropriate operation is always available in the domain ontology. For more complex services, e.g. with additional (compared to the template operation) parameters or functionality, it might become necessary to combine several operations from the domain ontology.
- Our current focus on describing only inputs and outputs of services can lead to low precision [11]. Therefore, preconditions and effects will also be included into the descriptions.

The adapted procedure will then be integrated into an architecture for discovery and retrieval of geographic information [12].

Finally, the reasoning procedures offered by tools such as the SSM will be examined more closely. E.g., it might be sensible to extend the matching algorithm from only considering input parameters that are more general to also acknowledging concepts that are more specific. This might be helpful if a requester has only a vague idea of what she is looking for.

Acknowledgements

The work presented in this paper has been supported by the European Commission through the ACE-GIS project

(grant number IST-2002-37724) and the BMBF as part of the GEOTECHNOLOGIEN program (grant number 03F0369A). It can be referenced as publication no. GEOTECH-64.

References

- [1] M. Klein and B. König-Ries, "A Process and a Tool for Creating Service Descriptions based on DAML-S", presented at 4th VLDB Workshop on Technologies for E-Services (TES'03), Berlin, 2003.
- [2] T. Kawamura, J.-A. De Blasio, T. Hasegawa, M. Paolucci, and K. Sycara, "Preliminary Report of Public Experiment of Semantic Service Matchmaker with UDDI Business Registry", presented at First International Conference on Service-Oriented Computing (ICSOC 2003), Trento, Italy, 2003.
- [3] E. Sirin, J. Hendler, and B. Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions", presented at "Web Services: Modeling, Architecture and Infrastructure" Workshop (in Conjunction with ICEIS2003), 2003.
- [4] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara, "Semantic Matching of Web Service Capabilities", presented at 1st International Semantic Web Conference (ISWC2002), Sardinia, Italy, 2002.
- [5] The DAML Services Coalition, "DAML-S: Web Service Description for the Semantic Web", presented at 1st International Semantic Web Conference (ISWC2002), Sardinia, Italy, 2002.
- [6] G. Antoniou and F. Van Harmelen, "Web Ontology Language: OWL", in *Handbook on Ontologies*, S. Staab and R. Studer, Eds.: Springer, 2003, pp. 67-92.
- [7] F. Baader and W. Nutt, "Basic Description Logics", in *The Description Logic Handbook. Theory, Implementation and Applications*, F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, Eds. Cambridge: Cambridge University Press, 2003, pp. 43-95.
- [8] ISO/TC-211, "Text for DIS 19107 Geographic information - Spatial Schema", International Organization for Standardization 2002.
- [9] ISO/TC-211, "Text for FDIS 19111 Geographic information - Spatial Referencing by Coordinates. Final Draft Version", International Organization for Standardization 2002.
- [10] K. Sycara, S. Widoff, M. Klusch, and J. Lu, "Larks: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace", presented at First International Joint Conference on Autonomous Agents and Multi-Agent Systems, Bologna, Italy, 2002.
- [11] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller, "Adding Semantics to Web Services Standards", presented at The SemanticWeb - ISWC 2003. 2nd International Semantic Web Conference (LNCS 2870), Sundial Resort, Sanibel Island, Florida, USA, 2003.
- [12] E. Klien, U. Einspanier, M. Lutz, and S. Hübner, "An Architecture for Ontology-Based Discovery and Retrieval of Geographic Information", presented at 7th Conference on Geographic Information Science (AGILE 2004), Heraklion, Greece, 2004.

Supporting Interface Integration with a Simple Ontology

Damien Conroy, Jim Buckley, Tony Cahill

Department of Computer Science and Information Systems, University of Limerick.

E-mail: {damien.conroy, jim.buckley, anthony.cahill}@ul.ie

Abstract

When an interface to a software component is changed it is likely that clients of that interface must adapt their invocations. Where parameters of those invocations represent quantities with dimensions there may be potential to recombine them. Analysis of possible combinations must ultimately be carried out by domain experts. This paper describes how simple representations of parameters and their dimensions may be quickly compared using logical operators in a structured search space so as to narrow the search for useful combinations. We consider these representations to constitute a simple ontology, simple in the sense that it supports only the relation 'is a constituent dimension of', but capable nonetheless of supporting an efficient narrowing of the search space.

1 Introduction

Composing a system from independent software components involves the task of aligning client invocations with service interfaces. While the alignment of invocations and interfaces with syntactic discrepancies may be achieved through casting or simple translations, overcoming discrepancies in the real-world concepts that are represented is a more difficult task. Detailed specifications of interfaces and associations with structured knowledge of the application domain are required in order to specify the necessary translations.

Structured knowledge of a domain may be represented in one or more ontologies. The term *ontology* as used here is informally defined as a collection of concepts and the relations between them. In information systems the term ontology may often refer to a piece of software implementing the services of such a collection, concept searching or matching for example[1].

2 An Illustrative Example

Software components provide services through interfaces. In this example a Java API is used as an example of a component interface. The same principles would apply to an electronic trader presenting a structured XML document, an invoice for example, as an interface.

Consider a method that determines whether or not a vehicle has broken the speed limit. Such a method might have the following signature if used between tollbooths on a motorway:

```
Boolean isOverLimit(Integer distance,  
                    Integer time)
```

Alternatively a computer attached to a radar detector might use a method with the following signature:

```
Boolean isOverLimit(Integer speed)
```

What must be done if the tollbooth system is to use the same 'isOverLimit' method as supplied with the radar detector? The parameters used in invoking the 'old' method interface must be recombined to suit the 'new' method interface.

3 Information Insufficiency

The type of mismatch that we call *information insufficiency* arises when there is no apparent one-to-one mapping between the concepts associated with the available and required parameters, as depicted in Table 1. Often however, the apparently missing information could be discovered, if the means by which it may be inferred from available information were known. In this example, the knowledge that a distance divided by the time taken to travel it produces a speed is enough to provide a likely solution.

The ability to deal with the apparent insufficiency is dependent upon knowledge of concepts and the relations between them. Such knowledge, held in an ontology for the domain, could be used to identify potential inferences.

Table 1. Information Insufficiency

Required	Available
Speed (Integer)	
	Distance (Integer)
	Time (Integer)

Table 2. SI Base Units

Base Quantity	Name	Symbol
Length	Metre	M
Mass	Kilogram	Kg
Time	Second	S
Electric current	Ampere	A
Thermodynamic temperature	Kelvin	K
Amount of substance	Mole	mol
Luminous intensity	Candela	Cd

4 A Simple Ontology based on Dimensional Analysis

One category of domain concepts to which inference is traditionally applied in physics is that of International Standard (SI) quantities. There are seven base SI quantities or dimensions as listed in Table 2. These are the fundamental quantities from which all others are derived[2]. The process of determining whether or not the results of equations are of a particular quantity is well known in physics as *dimensional analysis*.

The representation of dimensions in ontologies is not novel. Gruber, for example, provides Knowledge Interchange Format (KIF) specifications for dimensions such as length, time and length/time to illustrate principles of ontology design for knowledge sharing[3]. However, for the purpose of narrowing the search space of parameter combinations we use simpler representations based on vectors. Any base or derived quantity in the SI system may be represented using a vector of seven exponents[4]. For example, using the order of appearance in Table 2, *distance* would be represented with the following vector:

$$[1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

time⁻¹ would be represented by the following vector:

$$[0 \ 0 \ -1 \ 0 \ 0 \ 0 \ 0]$$

and *speed* would be represented using the vector (the vector sum of the previous two):

$$\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This set of vectors may be considered as a simple ontology, representing the SI concepts and the hierarchical "is a constituent dimension of" relationships between them. It may be used to identify potential inferences from combinations of the available parameters.

5 Combining Available Parameters

When searching for the required parameters several combinations of the available parameters must be considered. The process involves three steps:

Enumerate all subsets of available source parameters.

Eliminate all subsets that lack the necessary dimensions.

Examine the remaining subsets to determine if they can infer a missing parameter.

5.1 Enumeration

In the example the following possibilities must be considered when deriving speed from distance and time.

$$\begin{aligned} \{distance, time\} &\Rightarrow \{speed\} ? \\ \{distance\} &\Rightarrow \{speed\} ? \\ \{time\} &\Rightarrow \{speed\} ? \end{aligned}$$

Once it has been determined that speed can be inferred from a combination of distance and time it makes sense to re-examine the ontology to determine whether or not it could be inferred from a smaller subset of the available parameters, providing a translation with less redundancy. The sets {distance}, {time} and {distance, time} along with the empty set, {}, which we shall ignore, are all subsets of the set {distance, time}. Together they constitute the powerset, that is the set of all subsets, of {distance, time}. A powerset of an n-element set contains 2ⁿ elements.

If the set of available parameters has two elements it can be seen that, ignoring the empty parameter set, there are three (2² - 1) sets to be considered as the potential source for the derivation of a missing parameter. If there were *m* missing parameters then (2ⁿ - 1) * *m* possibilities exist for inference.

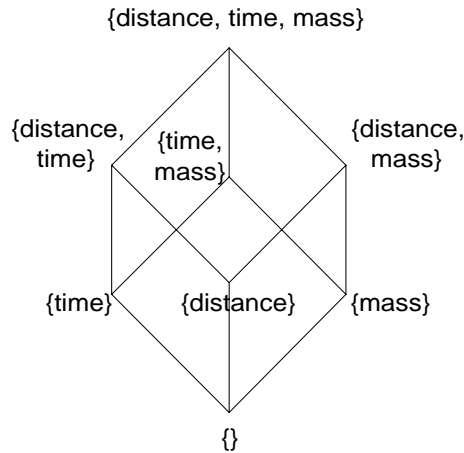


Figure 1. Lattice representation of the powerset of {distance, time, mass}.

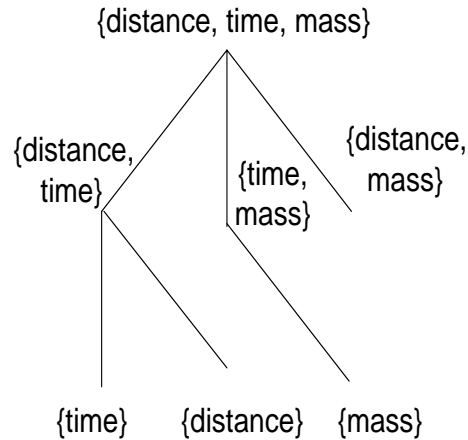


Figure 2. Tree representation of the powerset of {distance, time, mass}.

5.2 Elimination

Eliminating 'non-starters' is a simpler problem than that of determining whether or not the set of available parameters (which shall be referred to as source parameters) can be used to infer the set of missing parameters (which shall be referred to as target parameters.)

In the example the vectors of the ontology may be simplified to binary numbers where a particular bit is set if a particular dimension is present (regardless of the exponent) and unset if it is not. This would give the binary number 1010000 for the vector sum of the sources, and the binary number 1010000 for the target vector. If the result of a bitwise AND between the source binary number and the target binary number results in anything other than the target binary number itself then the "has the same dimensions as" relation does not exist between them. If that relation does not exist between them then the source parameter set provides no potential to infer that target parameter. A set of source parameters may be ruled out if it provides no potential inference for any of the target parameters. Otherwise each of its subsets must be similarly analysed to identify a minimal set. Consequently, it is beneficial to examine larger source parameter sets earlier in the elimination. This implies that parameter subsets should be processed in an order imposed by the subset inclusion operator. The powerset of a set partially ordered by the subset operator may be represented as a lattice, as shown in Figure 1. Lattices are similarly used in data mining applications to mine multidimensional association rules[5].

Here the lattice is modified to produce a tree (Figure 2) where each node represents a subset of the source parameter

set. The tree is traversed depth-first recursively. Starting at the top of the tree, when a node is visited, the binary number representing the combined dimensions of its parameter set is compared using a bitwise AND with the binary number of each target parameter to test the "has the same dimensions as" relation. If a node cannot be ruled out then its child nodes are visited. The nodes that are not ruled out are potential sources for useful inferences.

5.3 Evaluation

Potential inferences must be presented to a domain expert for final evaluation. Once the correct inferences have been identified they may be used as a basis for the generation of translations. Translations may be implemented with 'glue code'[6]. In the given example a skeleton method accepting a distance and time as parameters and providing a speed as a return value might be generated. In other cases integration experts might develop stylesheets for the translation of structured documents using the same principles.

6 Current Work

Current work involves expanding the set of domain types under consideration beyond the SI quantities, looking at business document elements and ebXML, the electronic business markup language, in particular[7]. ebXML provides basic core components, analogous to basic quantities and aggregate components, analogous to derived quantities. While these basic and aggregate core components may adequately describe domain entities they do not describe any relation other than aggregation, an address has a state and

city for example, but a city may not necessarily have a state associated with it.

The vector representations described are not expressive enough to depict the relations between such application domain concepts. Neither do they support the representation of properties other than *"has the constituent dimension"*. Application domain concepts necessitate richer representations and we are using the Web Ontology Language (OWL), through the Jena API to build these[8][9]. As a first cut we used OilEd to build the SI quantity ontology described above using OWL rather than vectors[10]. This proved to be an interesting exercise as OWL allows the introduction of properties such as units of representation and conversion factors that could not be described or associated using the vectors. Currently we are moving some of the ebXML domain concepts into OWL-based ontologies and adding extra properties, allowing us to infer a missing 'state' parameter where the 'city' is provided, for example.

7 Conclusion

In reconciling component interfaces the final evaluation of potential inferences and translations rests with a domain expert. However, much can be done to help the domain expert focus on the set of most promising combinations of available information. A suitable ontology for domain knowledge, coupled with an ordering of the search space can rule out infeasible combinations at an early stage, and highlight the most promising possibilities.

Acknowledgements

This research is funded by Enterprise Ireland and QAD Ireland under the Innovation Partnership Programme IP/2001/006.

References

- [1] B. Smith, in "The Blackwell Guide to the Philosophy of Computing and Information", Blackwell Publishers, 2003.
- [2] A. Bhagwat, "Applying Dimensional Analysis to Business Intelligence Systems", The Data Administration Newsletter (TDAN.com), <http://www.tdan.com/i020ht04.htm>
- [3] T. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", in "Formal Ontology in Conceptual Analysis and Knowledge Representation", Kluwer Academic Publishers, 1993.
- [4] G. Novak, "Conversion of Units of Measurement", IEEE Transactions on Software Engineering v.21 n.8, p.651-661, 1995.
- [5] J. Han and M. Kamber, "Data Mining: Concepts and Techniques", Morgan Kaufmann Academic Press, 2001.
- [6] M. Burstein, D. McDermott, D. Smith and S. Westfold, "Derivation of glue code for agent interoperation", in Proc. of the 4th Int'l. Conf. on Autonomous Agents, pages 277-284, 2000.
- [7] ebXML Enabling a Global Electronic Market, Available at: <http://www.ebxml.org>
- [8] OWL Web Ontology Language Overview, Available at: <http://www.w3.org/TR/2003/PR-owl-features-20031215/>
- [9] Jena 2 - A Semantic Web Framework, Available at: <http://www.hpl.hp.com/semweb/jena2.htm>
- [10] OilEd, Available at: <http://oiled.man.ac.uk/>

SWETO: Large-Scale Semantic Web Test-bed

Boanerges Aleman-Meza, Chris Halaschek, Amit Sheth, I. Budak Arpinar, Gowtham Sannapareddy
Large Scale Distributed Information Systems (LSDIS) Lab
Computer Science Department, University of Georgia
Athens, GA 30602-7404
{boanerg, ch, amit, budak}@cs.uga.edu, gowtham@uga.edu

Abstract. The emergent Semantic Web community needs a common infrastructure for testing the scalability and quality of new techniques and software which use machine processable data. Since ontologies are a centerpiece of most approaches, we believe that for an accurate evaluation of tools for quality, scalability and performance, the research community needs a freely available ontology with a large description base. If the use of tools is to be for advanced semantic applications, such as those in business intelligence and national security, then instances in the knowledge base should be highly interconnected. Thus, we propose and describe a Semantic Web Technology evaluation Ontology (SWETO) test-bed. In particular, we address the requirements of a test-bed to support research in semantic analytics, as well as the steps in its development, including, ontology creation, semi-automatic data extraction, and entity disambiguation.

1. Introduction

Considering that there are somewhere between 20 to 50 ontology tools alone [16, 17], the question arises: how do we test and compare them? Similarly, applications that utilize ontologies for inference, semantic integration, and semantic analytics, require a benchmark for quality, scalability and performance evaluations. Thus, the emergent Semantic Web community needs a common infrastructure for both testing and evaluations. In particular, we feel there is a need to have a large, high quality test ontology from which various ontology tools can assess and test their scalability and other properties.

Of particular interest is not just the schema of the ontology, but also the population (instances, assertions or description base) of the ontology. A highly populated ontology (ontology with instances or assertions) is critical for assessing effectiveness, and scalability of core semantic techniques such as semantic disambiguation, reasoning, and discovery techniques. Ontology population has been identified as a key enabler of practical semantic

applications in industry; for example, Semagix¹ reports that its typical commercially developed ontologies have over one million objects [18]. So far, such ontologies have not been available to the research community.

Another important factor related to the population of the ontology is that it should be possible to capture instances that are highly connected (i.e., the knowledge base should be deep with many explicit relationships among the instances). This will allow for a more detailed analysis of current and future semantic tools and applications, especially those that exploit the way in which instances are related. This is exemplified in our SemDis² project, in which new complex semantic relationships can be queried and discovered through traversing sequence of links among the instances of interest. Clearly, an ontology and corresponding knowledge base of real-world scale are needed as a benchmark for evaluating and comparing such tools and techniques.

To this end, we propose a Semantic Web Technology evaluation Ontology (SWETO³), that captures real world knowledge with over 40 classes populated with a growing set of relevant facts, currently at about one million instances. As part of the creation of SWETO, we have adopted the following iterative process that allows the periodic extension the ontology and its instances:

- (i) Designing SWETO schema using an ontology design toolkit (detailed later),
- (ii) Identifying knowledge sources that can be used to populate parts of SWETO without focusing on a specific domain,
- (iii) Utilizing extractors (written by humans using a toolkit) to periodically and automatically extract parts of knowledge from various open and public sources,
- (iv) Semi-automatically applying disambiguation techniques to extracted instances in the ontology (with

¹ <http://www.semagix.com>

² <http://lsdis.cs.uga.edu/Projects/SemDis/>

³ <http://lsdis.cs.uga.edu/Projects/SemDis/Sweto/>

limited human involvement) to eliminate redundancies and improve quality of the knowledge base,

(v) Providing capabilities for exporting SWETO and its instances from an internal representation to World Wide Web Consortium (W3C) standards, namely either OWL [13] or RDF [14]; thus allowing open use of SWETO.

The remaining sections of this paper are organized as follows: Section 2 details related work in this area; Section 3 describes the overall methodology of our approach for creation of SWETO; Section 4 presents the current results of our work; Section 5 provides conclusions and some future directions for SWETO.

2. Related Work

Due to the infancy of the Semantic Web, little research has been focused on the development of an evaluation benchmark or test-bed for it. One current and ongoing effort however is TAP [2], which provides a large knowledge base annotated using RDF and is described as a "... shallow but broad knowledge base ..." [2]. Our work differs in that we provide a smaller schema, but with a much larger number of instances that are highly interconnected. Additionally, we provide the option to serialize the ontology using OWL, allowing for more constraints and expressiveness at the schema level.

3. Methodology

SWETO is an ontology that incorporates instances extracted from heterogeneous sources. Automatic population is created by extractors (detailed in Section 3.3).

3.1. Ontology Creation

The test-bed has been created in a bottom-up fashion where the data sources dictate the classes and relationships defined in the ontology, similar in spirit to the concept of emergent semantics [1, 15].

To illustrate with an example, consider the listing of "people" in a computer science department. Typically, they would be listed separately as Faculty, Students and Staff. In such cases we create appropriate classes in the ontology and populate them with instances.

In SWETO, the ontology was created using Semagix Freedom, a commercial product which evolved from the LSDIS lab's past research in semantic interoperability and the SCORE technology [6]. The Freedom toolkit allows for the creation of an ontology, in which a user can define classes and the relationships that it is involved in using a graphical environment.

3.2. Selection of Data Sources

Creation of a solid test-bed requires meticulous selection of data sources. We focused our selection of data sources by considering the following factors:

(i) Selecting sources which were highly reliable Web sites that provide instances in a semi-structured format, unstructured data with parse-able structures (e.g., html pages with tables), or dynamic web sites with database back-ends. In addition, the Freedom toolkit has useful capabilities for focused crawling by exploiting the structure of Web pages and directories.

(ii) We carefully considered the types and quantity of relationships available in a data source. Therefore we preferred sources in which instances were interconnected.

(iii) We considered sources whose instances would have rich metadata. For example, for a 'Person' instance, the data source also provides attributes such as gender, address, place of birth, etc.

(iv) Public and open sources were preferred, such as government Web sites, academic sources, etc. because of our desire to make SWETO openly available.

3.3. Knowledge Extraction

In SWETO, all knowledge (or facts that populate the ontology) is extracted using Semagix Freedom software. Essentially, extractors are created within the Freedom environment, in which regular expressions are written to extract text from standard html, semi-structured (XML), and database-driven Web pages. As the Web pages are 'scraped' and analyzed (e.g., for name spotting [19]) by the Freedom extractors, the extracted instances are stored in the appropriate classes in the ontology. Additionally, provenance information, including source, time and date of extraction, etc., is maintained for all extracted data. We later utilize Freedom's API for exporting both the ontology and its instances in either RDF [14] or OWL [13] syntax. For keeping the knowledge base up to date, the extractors can be scheduled to rerun at user specified time and date intervals.

Automatic data extraction and insertion into a knowledge base also raise issues related to the highly researched area of entity disambiguation [7, 8, 9, 10]. In SWETO, we have focused greatly on this aspect of ontology population. Using Freedom, instances can be disambiguated using syntactic matches and similarities (aliases), customizable ranking rules, and relationship similarities among instances. Freedom is thus able to automatically disambiguate instances as they are extracted [6].

Furthermore, if Freedom detects ambiguity among new instances and those within the knowledge base, yet it is unable to disambiguate them within a preset degree of certainty, the instances are flagged for manual

Lastly, there are special cases in which neither the software, nor humans can directly determine if two instances are the same. For example, consider two persons named ‘John Smith’. Without metadata attributes, neither the system nor humans can determine what to do by only looking at the instance name. This is a future research direction we wish to follow in which semantic similarity can be used to state with some degree of certainty that these two persons (i.e. ‘John Smith’), are in fact the same person. For now, we remove these types of instances from the knowledge base in order to maintain both cleanliness and consistency.

Our aim of achieving a test-bed of over 1 million instances is near completion. The current population includes over 800,000 instances and over 1,500,000 explicit relationships among them. Here we provide initial statistics that illustrate the size in terms of instances and relationships connecting them.

Table 1. SWETO test-bed ontology initial metrics

What makes this work more valuable is in respect to how inter-connected the instances are (this currently is not available in a taxonomy and in most current ontologies that are freely available). As mentioned earlier interconnectedness becomes critical in semantics analytics applications (such as [3]). Table 2 summarizes a subset of the relationships connecting instances in the ontology. Note that some relationships apply to a variety of types of instances, such as the “located in” relation.

Subset of relationships	# Explicit relations
located in	30,809
responsible for (event)	1,425
Listed author in	1,045,719
(paper) published in	467,367

Table 3. SWETO statistics on disambiguation

In addition, SWETO homepage provides more details (<http://lsdis.cs.uga.edu/proj/Sweto/>). There, we provide a graphical user interface for browsing of SWETO ontology (through the use of Touchgraph⁴) as illustrated in Figure 1, the latest version of the knowledge base (instances), our own native API for easy use (alternately tools such as Jena [12] could be used), and a detailed description of the data sources. Currently, SWETO is also being used to support and evaluate provenance and trust research at UMBC, and it is being used by Semagix to evaluate effectiveness and performance of semantic metadata extraction and enhancement technology.



In this paper, we presented SWETO, a test-bed for testing effectiveness and scalability of current and future semantic Web applications and techniques.

Our research with SWETO test-bed has primarily been driven by the discovery of semantic associations [4] and their ranking [5]. Therefore, we aim for continuing the

⁴ <http://www.touchgraph.com/>

population of the ontology by further inter-connecting instances in order to provide a diverse test-bed for testing semantics analytics research ideas.

As mentioned in Section 3.3, we also wish to further investigate the use of semantic similarity for entity disambiguation.

6. Acknowledgements

SWETO test-bed is an effort that incorporated ideas and suggestions from different people in the LSDIS lab to whom we are thankful. Additionally, we would like to acknowledge our UMBC collaborators, especially Tim Finin, Anupam Joshi, and Li Ding who we are jointly working with on the SemDis project.

We also thank Semagix, Inc. for providing its Freedom product. In particular, we would like to especially thank David Avant and Yashodhan Warke for their insightful comments and reviews.

This work is funded in part by National Science Foundation (NSF) Awards 0219649 ("Semantic Association Identification and Knowledge Discovery for National Security Applications") and IIS-0325464 ("SemDis: Discovering Complex Relationships in Semantic Web"). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

- [1] S. Staab: Emergent Semantics. IEEE Intelligent Systems 17(1), 2002. pp. 78-86
- [2] R. Guha and R. McCool, "Tap: A Semantic Web Test-Bed", Journal of Web Semantics, 1(1), Dec. 2003, pp. 81-87
- [3] A. Sheth, B. Aleman-Meza, I. B. Arpinar, C. Halaschek, C. Ramakrishnan, C. Bertram, Y. Warke, D. Avant, F. S. Arpinar, K. Anyanwu, and K. Kochut, [Semantic Association Identification and Knowledge Discovery for National Security Applications](#), Special Issue of Journal of Database Management on Database Technology for Enhancing National Security, Eds: L. Zhou and W. Kim, 2004 (Accepted).
- [4] K. Anyanwu, and A. Sheth. [r-Queries: Enabling Querying for Semantic Associations on the Semantic Web](#). Twelfth International World Wide Web Conference, Budapest, Hungary. May 20-24, 2003; pp. 690-699
- [5] B. Aleman-Meza, C. Halaschek, I. B. Arpinar, and A. Sheth, [Context-Aware Semantic Association Ranking](#), First International Workshop on Semantic Web and Databases, Berlin, Germany, September 7-8, 2003; pp. 33-50
- [6] A. Sheth, C. Bertram, D. Avant, B. Hammond, K. Kochut, and Y. Warke. (2002). [Managing semantic content for the Web](#). IEEE Internet Computing, 6(4), 2002. pp 80-87
- [7] R. Mihalcea, and S. I. Mihalcea: Word Semantics for Information Retrieval: Moving One Step Closer to the Semantic Web. ICTAI 2001: 280-287.
- [8] P. Resnik, "Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language", Journal of Artificial Intelligence Research, 1999.
- [9] V. Kashyap, and A. P. Sheth, [Semantic and schematic similarities between database objects: A context-based approach](#). VLDB Journal, 5(4):276—304, 1996.
- [10] M. Rodriguez, and M. Egenhofer, Determining Semantic Similarity among Entity Classes from Different Ontologies, IEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 2, March/April 2003.
- [11] S. Handschuh, S. Staab. CREAM - CREATing Metadata for the Semantic Web. Computer Networks. 42, pp. 579-598, Elsevier 2003.
- [12] B. McBride. Jena: A semantic Web toolkit. IEEE Internet Computing, 6(6), 55-59, 2002.
- [13] S. Bechhofer, F. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, et al. (2003). OWL Web Ontology Language Reference. W3C Proposed Recommendation, from <http://www.w3.org/TR/owl-ref/>
- [14] O. Lassila, & R. Swick. (1999). Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation, from <http://www.w3.org/TR/REC-rdf-syntax/>
- [15] V. Kashyap and C. Behrens. "The Emergent Semantic Web: A Consensus approach for Deriving Semantic Knowledge on the Web", Proceedings of the International Semantic Web Working Symposium, July 2001, Stanford, USA.
- [16] M. Denny. "Ontology Building: A Survey of Editing Tools", available at <http://www.xml.com/pub/a/2002/11/06/ontologies.html>
- [17] "A survey on ontology tools." OntoWeb Consortium, 2002. http://www.aifb.uni-karlsruhe.de/WBS/ysu/publications/OntoWeb_Del_1-3.pdf
- [18] A. Sheth, C. Ramakrishnan. [Semantic \(Web\) Technology In Action: Ontology Driven Information Systems for Search, Integration and Analysis](#). IEEE Data Engineering Bulletin, Special issue on Making the Semantic Web Real, 26(4), pp. 40-48, 2003.
- [19] B. Hammond, A. Sheth, and K. Kochut. [Semantic Enhancement Engine: A Modular Document Enhancement Platform for Semantic Applications over Heterogeneous Content](#). In V. Kashyap & L. Shklar (Eds.), Real World Semantic Web Applications (pp. 29-49): Ios Pr Inc. 2002.

Towards Ontological Modelling of Historical Documents

Vanessa Mirzaee*, Lee Iverson*, Babak Hamidzadeh[†]

*Dept. of ECE, University of British Columbia, Vancouver BC

[†]Boeing Corporation, Seattle WA

vanesam@ece.ubc.ca, leei@ece.ubc.ca, babak.hamidzadeh@boeing.com

Abstract. In this paper we describe a methodology we have adopted for coding the semantic structure of a historical document and the resulting semantic model. To do this, we adapted currently available methodologies for ontology engineering to the context of semantic document coding. Using Protégé-2000 we then used this methodology to develop a formal ontological model and finally to encode a historical document covering the evolution of the constitution of modern Iran. The resulting semantic model was then evaluated by direct reference back to the set of competency questions and motivating scenarios used to develop the model. Our implementation was successful in answering these competency questions as well as in providing support for the selected scenarios. The implementation and the evaluation results are presented along with our proposed future work.

1. Introduction

Until recently, it has been assumed that the main advantage of electronic formats over printed matter is the convenience of being able to find the material without having to physically obtain it from a library or other repository [9]. However, once we have this information in a digital format, it is unclear as to how the user might interact with it besides being able to print it and/or read it. We believe that digital documents have the potential to provide us with more functionality than traditional printed matter does.

In particular, we have chosen to use an ontological approach to code documents, thus allowing a community to (1) *share and reuse* their knowledge, (2) *capture the semantics* implicit in the documents, and (3) allow *computational manipulation* of the acquired knowledge. This manipulation of the document's meaning would allow automatic reasoning beyond the simple queries and keyword search provided by current information retrieval methods.

Consider the case of historical document archives. A wealth of historical information is now available in digital form through different resources such as digital libraries. These digital media usually integrate meta-data that

provides some information about their content [4]. These collections provide the ability to retrieve the best-matched documents for any search request. The field of information retrieval for these kinds of document collections is an active area of research [6].

Instead of basing these searches on keywords, it would be ideal for electronic historical archives to provide methods and techniques of posing and resolving historical questions and then providing access to the sources of the claims used to resolve them. For example a historian will want to query relationships between characters, institutions, events, and locations of these events. Significantly, it is vital to capture how these relationships change over time.

To ground our work, we have chosen to examine a particular historical document, "*History of the Iranian Constitutional Revolution*," [8] describing the evolution of the modern state of Iran over a 50-year period. We suggest that the methodology adopted will apply equally well to a similar class of documents and that our ontological model will generalize to any document which records historical events in a similar manner.

Thus, we have presented an approach to representing the knowledge within a historical document that allows such sharing, reuse and automatic reasoning by capturing its semantic content using ontologies. Next we will present the methodology used to build an ontological model to represent the knowledge found in a historical document. We show this with reference to our example ontology.

2. Methodology and Implementation

Building a well-developed, usable, and sharable ontology represents a significant challenge. There is great diversity in the way ontologies are designed as well as in the way they try to represent the world.

A range of methods and techniques have been reported in the literature regarding ontology building methodologies. However, there is ongoing argument within the ontology community about the best method to build them [13; 9; 2]. Given that the knowledge to be

captured usually depends critically on a combination of the domain and the applications being designed to exploit this knowledge [14], it is no surprise that these methodologies are primarily inspired by enterprise modeling or software engineering. For our purposes, it was important to scale them down and adapt them to facilitate document coding.

We divide the ontology building process into the following stages:

1. Identifying the purpose, scope, and users
2. Domain analysis and knowledge acquisition
3. Building a conceptual (informal) ontology model
4. Formalization
5. Evaluation

In our method, we focus on an evolving prototype of the ontology. In this model, every step forms part of a process to evolve the prototype. Moreover, at each stage, it is possible to go back to any previous stage of the development process, in order to satisfy emerging requirements. This makes the evolving prototype useful for developing any ontology from scratch. Figure 1 illustrates how these steps are related, and in what order they can be performed to complete the ontology building process. We make every effort to maintain the following criteria for each and every stage of the development process: Clarity; Coherence; Extensibility; Minimal encoding bias; and Minimal ontological commitment [5; 19].

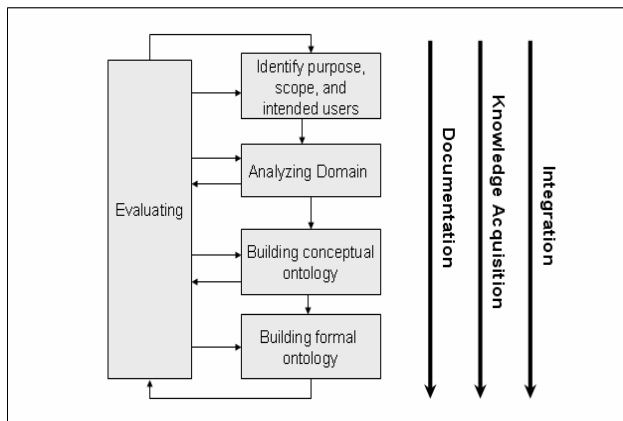


Figure 1 Our ontology development process. Integration, Knowledge Acquisition, and Documentation are carried out throughout the entire development process.

2.1. Identify purpose, scope, and intended users

The main purpose for building this ontology is to capture the semantics of a historical document, especially the temporal and dynamic aspects of the concepts and their interrelations. To promote sharing, reuse and enable better integration with existing knowledge sources we relied heavily on the consensual terminology available in general ontologies. The selected audience included both

the general public and historians and biographers who might directly access the semantic models.

The requirements gathered were formulated as a set of competency questions and motivating scenarios that our model must answer and provide support for. A few of these competency questions are presented in Table 1.

Table 1 Some Competency Questions

1. Who was <i>Person P</i> ?
2. In What <i>Events</i> was <i>Person P</i> involved?
3. What <i>Positions</i> did <i>Person P</i> hold?
4. When did <i>Person P</i> hold these <i>Positions</i> ?
5. Who was taking over <i>Person P</i> 's <i>Position Po</i> ?
6. What was the governmental position hierarchy at the time <i>Person P</i> held <i>Position Po</i> ?

2.2. Domain analysis and knowledge acquisition

Using the competency questions and scenarios we then produce a set of concepts and terms covering the full range of information that the ontology must characterize to satisfy these requirements. In this phase, we use knowledge acquisition techniques such as brainstorming, in conjunction with formal analysis of the text to gather all potential relevant terms into a glossary [3].

Table 2 Partial view of Glossary of Terms

Term	Definition	Resource
Person	An individual, someone, somebody. An agent with certain rights and responsibilities and the ability to reason, deliberate, make plans, etc. This is essentially the legal/ethical notion of a person.	WordNet & SUMO
Government	A ruling body of a country	Oxford Dictionary
Position	A formal position of responsibility within an organization.	SUMO & Merriam Webster
Governmental Position	A formal position of responsibility within a governmental organization	SUMO

This glossary includes the terms, their definition or description, and may include additional information, such as examples that help understanding these definitions. In order to provide definitions for the terms, we consulted dictionaries such as the Merriam Webster Dictionary and the Oxford Dictionary as well as general purpose ontologies such as SUMO [18], and WordNet [20]. Table 2 shows a partial view of our glossary of terms.

2.3. Building an informal ontological model

Once we have a relatively complete glossary of terms, we identify concepts, relations within the concepts, and their

attributes. We use the guideline provided in [14] to do so. The results are stored in document tables called the Concept Dictionaries [3]. At this stage, the concepts are structured into naturally occurring groups using a combination of the approaches introduced in [14] and [10]. For example, concepts most related to one another are placed within the same Concept dictionary. We categorized our concepts into five concept dictionaries relating to *people*, *places*, *events*, *documents*, and *time*. Each of these categories holds the concepts that are most related

For the next step, we use the previously generated concept dictionaries, along with the motivating scenarios and a middle-out approach to develop our graphical conceptual ontology model. The middle-out approach operates by identifying the most important concepts first and then generalizing or specializing these concepts within the group from that point [14]. Our conceptual model not only represents the concept taxonomy but also the other (non-taxonomic) relations that hold amongst the concepts within our domain.

Throughout the ontology building stages, we queried existing ontology libraries, such as Ontolingua [15] and SUMO [18] to search for similar or related terms and relations that might be useful. This was done in order to speed up the development process as well as to gain a better insight of how to build a particular area or set of concepts within our ontology. Thus we were able to build our ontology on a well-grounded structure. In particular, the time concepts were derived from general time ontologies [15; 18; 21] and the temporal relations in TELOS [12]. Events were based on Sowa's thematic roles or case relations [17]. Places were defined using standard ontologies for geographic information representation and categorization [1; 7; 11].

Figure 2 shows the top-level concepts in our domain. We identified five central concepts within our ontology: AGENT, PLACE, EVENT, DOCUMENT, and TIME. Every other concept in this domain is defined around these primitive concepts.

An important characteristic of the proposed ontological model is its capability to represent temporally dynamic concepts. This is of particular importance for historical data since the concepts and the relations between them change and evolve through time. This is accomplished by associating a time interval with each relation, as was done in Telos [12]. Additionally, this model not only captures the relationships between the concepts but also demonstrates the interrelated hierarchal structure within them. An example of such hierarchical structures found within our document is the governmental position hierarchy. In this hierarchy, not only do the people that hold positions change but the structure itself evolves throughout time.

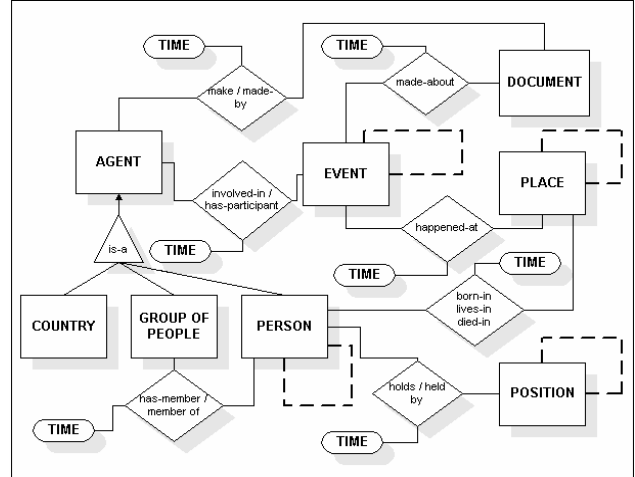


Figure 2 Overview of main concepts and relations in our history ontology. Dotted-lines denote the existence of type reflexive relations within a concept. The time tag on a relation indicates that a particular relation is time-dependent.

2.4. Building a formal ontological model

The next step in our approach was to build a formal ontology based on the conceptual model. After a review of available ontology development environments, we selected Protégé-2000 [16] to formalize and instantiate our ontology. Our selection was based on the tool's expressiveness, flexibility, customizability, scalability, extensibility, and usability. Significantly, it also provided us with the facilities to test and evaluate our model.

Additionally, Protégé-2000 provides facilities to impose constraints to concepts and relations. While creating the ontology, it is necessary to make general assertions about fundamental concepts, and be able to later test and ensure these assertions hold across the entire knowledge-base. For example, in our ontology it was useful to assert common-sense constraints such as:

- All instances of Person have exactly one birth-date.
- A Person's birth-date must precede the death-date.
- Every Event in which a Person is involved, must take place between his or her birth-date and death-date.
- For any given time interval there can only be one person holding the position "king".

2.5. Evaluation

After designing, building, and formalizing our ontology using Protégé and enforcing constraints on attributes and relations, we used the knowledge acquisition forms provided in Protégé to instantiate our history ontology.

Over seven hundred and fifty (750) instances were extracted from the history book and included in our ontology. Amongst these instances we find people, places, documents, and events.

In order to evaluate the correctness and completeness of the created ontology, we use the query and visualization facilities provided by Protégé. We use the built-in query engine in Protégé for the simple query searches and use additional query plug-in provided in Protégé to create more sophisticated searches. We also use Protégé-visualization plug-ins to browse the ontology and ensure its consistency. Visualization aids were particularly helpful when trying to understand hierarchical relations.

3. Conclusions and Future Work

In this work we confronted the limitations of traditional electronic documents. In particular we were interested in capturing the semantics of a historical document to allow for richer retrieval, reuse and manipulation of its embedded knowledge than what is capable with standard text manipulation tools.

After adapting existing methodologies to the problem of text coding, we developed an ontology motivated by historical and biographical needs and the contents of the book "*History of the Iranian Constitutional Revolution*." Using this ontology, we then encoded the book's claims and verified our coding and ontology by proving that the ontology allowed us to answer all of our category questions. Our implementation allowed us to get an overview of the general concepts in this book, relationships amongst these concepts and provided us with different methods for visualizing dynamic hierarchical structures of both governmental positions and geopolitical interdependencies. Additionally, this model captures the changes that these relations undergo through time (dynamicity). The temporal aspects of the knowledge we captured proved to be useful in making our representation more accurate and realistic.

In order to facilitate the utilization of models such as the one developed here, we will require applications that facilitate interacting with this information. One challenge will be to develop easy, intuitive interfaces to both access and query these models that will allow both sophisticated and naïve users to take advantage of the information they encode. In addition, we hope to develop ontology development tools that reflect the methodology developed and facilitate its application to new domains.

References

[1] Alani, H., Jones, C. and Tudhope, D. (2000). "Ontology-Driven Geographical Information Retrieval." GIScience 2000.

[2] Beck, H. and Pinto, H. S. (2003). "Overview of Approach, Methodologies, Standards, and Tools for Ontologies." The Agricultural Ontology Service (UN FAO).

[3] Blazquez, M., Lopez, M. F., Perez, A. G. and Juristo, N. (1998). "Building Ontologies at the Knowledge Level Using the Ontology Design Environment." In Proceedings of KAW'98, Banff, Canada.

[4] Burchardt, J. (2001). "Archiving the Internet - how to collect historical sources for the future." International Conference of the Association for History and Computing, Poznan, Poland.

[5] Gruber, T. R. (1995). "Toward principles for the design of ontologies used for knowledge sharing." International Journal of Human-Computer Studies, 43(5-6), 907-928.

[6] Hockey, S. M. (2000). "Electronic texts in the humanities: principles and practice." Oxford; New York. Oxford University Press.

[7] Islam, A. S., Bermudez, L. and Piasecki, M. (2003) Ontology for Geographic Information-Metadata (ISO-19115) <http://loki.cae.drexel.edu/~wbs/ontology/>.

[8] Kasravi, A. (1940). "Hoistory of the Iranian Constitutional Revolution." Tehran. Amir Kabir Publications.

[9] Lopez, M. F. and Perez, A. G. (2002). "Overview and Analysis of Methodologies for Building Ontologies." Knowledge Engineering Review, 17(2), 129-156.

[10] Lopez, M. F., Perez, A. G., Sierra, J. P. and Sierra, A. P. (1999). "Building a Chemical Ontology Using Methontology and the Ontology Design Environment." Ieee Intelligent Systems & Their Applications, 14(1), 37-46.

[11] Mark, D. M., Skupin, A. and Smith, B. (2001). "Features, Objects, and other Things: Ontological Distinctions in the Geographic Domain." Conference On Spatial Information Theory (COSIT), Morro Bay, CA, USA.

[12] Mylopoulos, J., Borgida, A., Jarke, M. and Koubarakis, M. (1990). "Telos: Representing Knowledge About Information Systems." ACM TOIS. 325-362.

[13] Noy, N. F. and Hafner, C. D. (1997). "The state of the art in ontology design - A survey and comparative review." AI Magazine, 18(3), 53-74.

[14] Noy, N. F. and McGuinness, D. L. (2001). "Ontology Development 101: a Guide to Creating Your First Ontology." Stanford, CA, Stanford University.

[15] Ontolingua. www.ksl.stanford.edu/software/ontolingua/ Knowledge System Labratory, Stanford University.

[16] Protege-2000 <http://protege.stanford.edu/index.html>.

[17] Sowa, J. F. (2000). "Knowledge Representation: Logical, Philosophical, and Computational Foundation." Pacific Grove, CA. Brooks Cole Publishing Co.

[18] SUMO (Suggested Upper Merged Ontology) <http://ontology.teknowledge.com/>.

[19] Swartout, B., Patil, R., Knight, K. and Russ, T. (1996). "Toward Distributed Use of Large-Scale Ontologies." Proceedings of KAW'96, Banff, Canada.

[20] WordNet <http://www.cogsci.princeton.edu/~wn/>.

[21] Zhou, Q. and Fikes, R. (2002). "A Reusable Time Ontology." Proceeding of the Ontologies for the Semantic Web Workshop, AAAI National Conference.

Applying Evidential Reasoning to Multiple Source Data Integration for Software Engineering Decision Support

George Shi, Reda Alhajj, Ken Barker

Department of Computer Science

The University of Calgary

2500 University Dr. NW, Calgary, AB, Canada T2N 1N4

{shig, alhajj, barker}@cpsc.ucalgary.ca

Abstract. In order to make optimal decisions during software engineering practices, we have to make use all data available, including lines of code, running time, performance matrix, market survey, benchmark testing, etc. Data from these sources may have different measurement scales and different qualities. Further some sources may provide incomplete data set. All these characteristics limit the use of traditional statistics method for data integration. This paper introduces basic concepts of evidential reasoning and proposes to integrate multiple data using evidential reasoning for supporting software engineering decision-making.

1. Introduction

Software Engineering decision-making such as build vs. buy, requirements prioritization, selection of COTS (commercial off the shelf) products, adoption of software process models, selection of business analysis models, is very crucial to the success of software development projects. There has been an increasing interest in software engineering decision support system (SE-DSS) from both industry and academia [1,2,3,4,5].

To provide support for decision making during software practices we have to make use all kinds of information available. For example, within an organization there is usually variety of data available such as data from software measurement, benchmark testing, end user survey, and market analysis. In order to make optimal decisions, it is better to use all these data. The reasons lie mainly in two-fold. A single source may not provide all the information required for decision-making. On the other hand, multiple source data may provide software engineering decision makers with information of higher accuracy and less uncertainty. Therefore we argue that the

integration of data from multiple sources plays very important role in implementing any SE-DSS.

Data from different sources are the results of qualitative or quantitative observations. There are variety of software engineering related data available in organizations: lines of code of a package, running time of a process, performance matrix for a web server, market survey for reporting products, benchmark testing for DBMS, just to name a few.

It was summarized [6] that there are four different measurement scales: nominal, ordinal, interval and ratio. Nominal data is simply a distinct category and serves only for labelling or naming the phenomenon. A typical example would be names of COTS product categories: OS, DBMS, Web Server, App Server, etc. Although numerical labels may be assigned to categories, the numerical values 1, 2, 3, ... n in nominal data are merely symbols. They cannot be manipulated mathematically. Ordinal data is the results of ranking measurements. For example, the results of performance test on a software product could be bad, fair, good and excellent. Data from different sources may also have different accuracies and completeness because of different ways of collection, manipulation and representation. All these characteristics make it difficult to apply traditional statistical methods to multiple source data integration.

Evidential Reasoning which is based on Evidential Theory [11] provides a heuristic scheme for handling data from multiple diverse sources. In fact it has shown great potential in other applications such as medical diagnosis [7], route planning [8], remote sensing classification [9] and geologic mapping [10]. However, there have been no publications found on applying Evidential Reasoning to multiple source data integration for software engineering decision support.

The remaining sections review the basic concept of Evidential Reasoning and introduce how to apply it to integrating multiple data sources for software engineering decision support. Some practical considerations for implementing Evidential Reasoning are also proposed.

2. Evidential reasoning

Mathematical theory of evidence proposed by Shafer [11] is an effective approach for data and/or knowledge integration. The theory also provides a mathematical framework for the description of incomplete knowledge. With this theory, a belief structure, along with mass function or basic probability assignment functions (BPA) provides a scheme for representing incomplete knowledge about a piece of evidence. Dempster's rule of combination then provides a tool to combine the total belief support from different sources. The interval between 'belief' and 'plausibility' (low and high probabilities) presents the uncertainty of the knowledge about the event.

Mass function or Basic Probability Assignment (BPA)

A set of mutually exclusive and exhaustive hypotheses is called a frame of discernment Θ

$$\Theta = \{h_1, h_2, h_3 \dots\}$$

Any subset of Θ is also a hypothesis. Beliefs can be assigned to all possible subset of Θ , denoted by 2^Θ . If the set is of size n , it will have 2^n subsets. The effect (support contribution) of each distinct evidence on the subset of Θ can be represented by a function called a *mass function*, or *basic probability assignment* (BPA). The mass function assigns a number in $[0, 1]$ to every subset of Θ such that the numbers sum to 1. That is,

$$\sum_{A \subset \Theta} m(A) = 1 \quad (1)$$

and

$$m(\Phi) = 0 \quad (2)$$

where $m(A)$ is the mass function and Φ is the empty set. In fact, a mass function is a quantitative representation of evidential support.

Belief function and plausibility function

Based on a mass function, a *belief function* (Bel) on hypothesis A is the sum of the mass function values of all subset of A :

$$Bel(A) = \sum_{B \subset A} m(B) \quad (3)$$

Therefore, $m(B)$ is the measure of the portion of total belief committed to hypothesis A , and $Bel(A)$ is a measure of the total amount of belief in hypothesis A .

A *plausibility function* of A , $Pls(A)$, is defined as

$$Pls(A) = 1 - Bel(\neg A) = \sum_{B \cap A \neq \Phi} m(B) \quad (4)$$

$\neg A$ denotes the complement of A . $Bel(\neg A)$ is the extent to which A has been refuted by the current evidence. $Bel(A)$ indicates amount of belief committed to A based on the given evidence, while $Pls(A)$ represents the maximum extent to which the current evidence allows one to believe in A . Both Bel and Pls can be used in decision rules for selecting the optimal hypothesis.

Belief interval

Usually $Bel(A) \neq Pls(A)$ and the true probability of A lies somewhere between Bel and Pls , which is often referred as the *belief interval* $[Bel, Pls]$.

One of the advantages of applying Evidential Reasoning over other reasoning methods is its ability to express ignorance. The commitment of belief to a subset A does not force the remaining belief to be committed to its complement (i.e. $Bel(A) + Bel(\neg A) \leq 1$). The amount of belief committed to neither A nor its complement $\neg A$ is the degree of ignorance which provides a measure of uncertainty.

Dempster's rule of combination

The central idea of evidential theory is the transformation of a large body of evidence (such as many sources) into manageable components. Dempster's rule of combination gives us the aggregation approach to combine different pieces of evidence to get a joint support and reduce the uncertainty at the same time.

Let Bel_1 , Bel_2 and m_1 , m_2 , denote two belief functions and their corresponding mass functions or BPAs. Dempster's rule defines a new mass function, denoted by $m_1 \oplus m_2$, which represents the combined support contribution of m_1 and m_2 over a subset of hypotheses:

$$m_1 \oplus m_2(H_k) = \frac{\sum_{A_i \cap B_j = H_k} m_1(A_i) m_2(B_j)}{1 - \sum_{A_i \cap B_j = \Phi} m_1(A_i) m_2(B_j)} \quad (5)$$

where H_k , A_i , $B_j \subset \Theta$ and $i, j, k = 1, 2, \dots, n$. Θ is a set of mutually exclusive and exhaustive hypotheses with size of n . The corresponding belief function $Bel_1 \oplus Bel_2$ may be computed from $m_1 \oplus m_2$ described in (5). Formula 5 is also called *orthogonal summation* of evidential support.

Condition

There is a condition to applying Dempster's rule of combination. It requires that two evidential sources be independent. The independence between sources in the evidential theory is different in meaning from the statistical independence. Two highly correlated sources of data may seem redundant in a statistical sense, but can improve our confidence on accuracies of decision-making results.

3. Data integration using Evidential Reasoning

Under the framework of the evidential theory, each individual data source can be considered as a piece of evidence. The value of each data source can be treated as a measurement for evaluating the evidential support committed to a set of hypotheses. In the context of software engineering decision-making, the study is mainly focused on the singleton case. That is, our purpose is to find out the degree of belief of each individual option (or hypothesis), for instance, choose a DBMS product for your backend data repository from a set of options:

$H = \{\text{Oracle, SQL Server, Sybase, MySQL}\}$. We would be more interested in finding $\text{Bel}(\text{Oracle})$, $\text{Bel}(\text{SQL Server})$, $\text{Bel}(\text{Sybase})$ and $\text{Bel}(\text{MySQL})$ than knowing $\text{Bel}(\text{Oracle, SQL Server})$ or $\text{Bel}(\text{Oracle, SQL Server, Sysbase})$, ...etc.

In the example shown in Figure 1, $m_{x_1}(H_i)$ is the mass function representing the degree of evidence X_1 in support of a given hypothesis H_i ($i = 1, 2, \dots, n$). For example, in SE-DSS, it represents the contribution from evidence X_1 supporting Option H_1 .

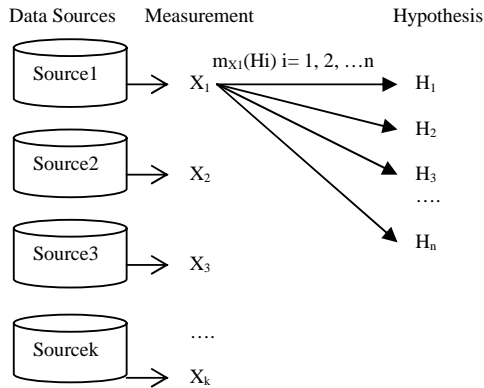


Figure 1. Evidence evaluation

In the singleton case, for the data source 1, formula (3) can be simplified as

$$\text{Bel}_{X_1}(B) = m_{X_1}(B)$$

For the example of COTS product selection, belief can be interpreted as the minimum amount of evidence supporting from a data source to an option.

Using Dempster's rule of combination we can then combines belief values from different data sources. The integrated evidential support is used to determine which hypothesis is the most optimal.

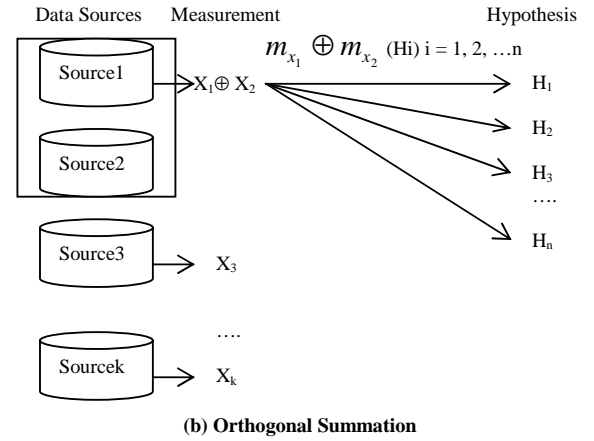
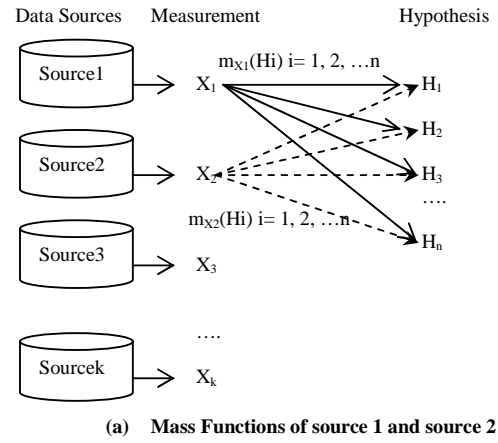


Figure 2. Dempster's rule of combination

Suppose m_{X_1} and m_{X_2} are two mass functions for source 1 and source 2, which are two independent bodies of evidences as shown in Figure 2-a. The combined mass function denoted as $m_{X_1} \oplus m_{X_2}$ can be computed using Formula (5) as shown in Figure 2-b.

We can treat $m_{x_1} \oplus m_{x_2}$ as a mass function and apply Dempsters' rule of combination to integrate it with m_{x_3} . Formula (5) can be used iteratively to combine all evidence support from different sources.

Because Dempster's rule is both commutative and associative, the order of combining multiple independent pieces of evidence does not affect the result. If the original pieces of evidence are independent, then the derived pieces of evidence are independent too [8].

An Example of applying combination rule

The aggregation of the evidential supports from each distinct data sources or evidence is achieved by orthogonal summation (Formula 5). It can be applied to any number of data sources.

Table 1. BPA values for a set of evidence from two different sources for selection of DBMS product

	Oracle	SQL Server	Sybase	MySql
Source 1	0.2	0.3	0.3	0.1
Source 2	0.1	0.3	0.4	0.0

To illustrate the procedure of orthogonal summation, suppose we want to select a DBMS product which would be best fit the system that is being developed. Here we

could think of two data sources available: for instance, one is about performance of query processing and transaction processing; the other is about security. The data was independently collected. BPA values in Table 1 indicate the degree of preference on the products. From table 1 you would notice that none of the row-wise sums equals 1. Taking source 1 as an example, the residual, $[1 - m_1(\text{Oracle}) - m_1(\text{SQLServer}) - m_1(\text{Sybase}) - m_1(\text{MySQL})]$ can be treated as the degree of ignorance (denoted as *Complement* in Table 2).

To calculate $m_1 \oplus m_2(H)$, where $H \subset \{\text{Oracle, SQL Server, Sybase, MySQL}\}$, we illustrate the procedure in Table 2. By integrating data from two sources (source 1 and source 2) based on Evidential Reasoning, we get total amount of belief for each of the product: $\text{Bel}_{m_1 \oplus m_2}(H)$. Based on Formula (4), the plausibility values are also calculated: $\text{Pls}_{m_1 \oplus m_2}(H)$. The evidential interval gives us a measurement for the uncertainty of preferences on a DBMS product.

Table 2 only shows an example with two data sources. As mentioned above, the Evidential Reasoning combination rule can be applied iteratively to any number of sources. The combined belief value and plausibility value can be used for selection of an optimal DBMS product.

Table 2. Orthogonal summation

Source 1 Source 2	Oracle 0.2	SQL Server 0.3	Sybase 0.3	MySQL 0.1	Complement 0.1
Oracle 0.1	Oracle $0.2 \times 0.1 = 0.02$	Φ $0.3 \times 0.1 = 0.03$	Φ $0.3 \times 0.1 = 0.03$	Φ $0.1 \times 0.1 = 0.01$	Oracle $0.1 \times 0.1 = 0.01$
SQL Server 0.3	Φ $0.2 \times 0.3 = 0.06$	SQL Server $0.3 \times 0.3 = 0.09$	Φ $0.3 \times 0.3 = 0.09$	Φ $0.1 \times 0.3 = 0.03$	SQL Server $0.1 \times 0.3 = 0.03$
Sybase 0.4	Φ $0.2 \times 0.4 = 0.08$	Φ $0.3 \times 0.4 = 0.12$	Sybase $0.3 \times 0.4 = 0.12$	Φ $0.1 \times 0.4 = 0.04$	Sybase $0.1 \times 0.4 = 0.04$
MySQL 0	Φ $0.2 \times 0 = 0$	Φ $0.3 \times 0 = 0$	Φ $0.3 \times 0 = 0$	MySQL $0.1 \times 0 = 0$	MySQL $0.1 \times 0 = 0$
Complement 0.2	Oracle $0.2 \times 0.2 = 0.04$	SQL Server $0.3 \times 0.2 = 0.06$	Sybase $0.3 \times 0.2 = 0.06$	MySQL $0.1 \times 0.2 = 0.02$	MySQL $0.1 \times 0.2 = 0.02$
	$\sum_{x \cap y = \text{Oracle}} m_1(x)m_2(y)$ =0.07	$\sum_{x \cap y = \text{SQLServer}} m_1(x)m_2(y)$ =0.18	$\sum_{x \cap y = \text{Sybase}} m_1(x)m_2(y)$ =0.22	$\sum_{x \cap y = \text{MySQL}} m_1(x)m_2(y)$ =0.02	$\sum_{x \cap y = \text{Complement}} m_1(x)m_2(y)$ =0.02
	$1 - \sum_{x \cap y = \Phi} m_1(x)m_2(y) = 1 - 0.49 = 0.51$				
$m_1 \oplus m_2(H)$	$0.07 / 0.51 = 0.14$	$0.18 / 0.51 = 0.35$	$0.22 / 0.51 = 0.43$	$0.02 / 0.51 = 0.04$	$0.02 / 0.51 = 0.04$
$m_1 \oplus m_2(\neg H)$	$0.35 + 0.43 + 0.04 = 0.82$	$0.14 + 0.43 + 0.04 = 0.61$	$0.14 + 0.35 + 0.04 = 0.53$	$0.14 + 0.35 + 0.43 = 0.92$	0.96
$\text{Bel}_{m_1 \oplus m_2}(H)$	0.14	0.35	0.43	0.04	0.04
$\text{Pls}_{m_1 \oplus m_2}(H) = 1 - m_1 \oplus m_2(\neg H)$	0.18	0.39	0.47	0.08	0.04

4. Practical considerations of applying Evidential Reasoning to multiple source data integration

The key to implementing Evidential Reasoning is to define mass function or BPA for a data source. Most implementations of Evidential Reasoning defined mass function values based on domain experts' knowledge. In fact, the representation of evidential support knowledge is actually a transformation of human experience, understanding and interpretation of nature into a computational domain.

Peddle [9] proposed a mass function definition approach based on frequency of data occurrence. In the applications of SE-DSS, the value of each data source indicates either a support or an objection to a set of hypotheses. Since we can treat these data sources as different pieces of evidence, the frequency of value occurrences within available data set such as historical software measurement data indicates the magnitude of support of the hypothesis to a certain extent. Using this method to represent the knowledge of evidential support can deal with data at all measurement scales. For the example of requirements prioritization, the priorities values of low, medium, high, highest may come from different stakeholders (difference data sources). The frequency of each value may be considered as evidential support to a requirement item.

After applying combination rule, we obtain an aggregated belief function over the frame of discernment for making a final decision. In statistic inference, usually there is only one choice in the decision-making process. With Evidential Reasoning, since an evidential interval bounded by support and plausibility is considered, there are a number of potential options to choose from to reach a decision. For example, the decision can be based on a maximum belief value, a maximum plausibility value or a maximum sum of belief and plausibility value.

5. Summary and Conclusions

The paper briefly introduced Evidential Reasoning. The focus was put on how to apply Evidential Reasoning to data integration for SE-DSS. An example was given on the details of calculating belief support from different data sources based on Dempster's rule of combination. Some practical considerations are also presented. The evidential reasoning does show its potential to data integration of

multiple sources for software engineering decision support although further study is needed for finding systematic ways of evaluation evidential support (mass function or BPA). Because data from different sources can be transformed into evidential belief domain, Evidential Reasoning can handle the different types of data. Uncertainty can also be measured through evidential interval.

References

- [1] G. Ruhe: Software Engineering Decision Support - Methodology and Applications. "Innovations in Decision Support Systems, International Series on Advanced Intelligence Volume 3, 2003, pp 143-174.
- [2] G. Ruhe: "Intelligent Support for Selection of COTS Products", Proceedings of the Net.ObjectDays 2002, Erfurt, Springer 2003, pp 34-45.
- [3] G. Ruhe: "Software Engineering Decision Support - A New Paradigm for Learning Software Organizations", Proceedings of the 4th Workshop on Learning Software Organizations, Chicago, Springer 2003, (p140).
- [4] D. Karlstrom, P. Runeson: "Decision Support for Extreme Programming Introduction and Practice Selection", Proceedings of the 14th Int. Conference on Software Engineering and Knowledge Engineering, 2002, Ischia, Italy.
- [5] D. M. Raffo, W. Harrison, J. Vandeville: "Software Process Decision Support: Making Process Tradeoffs Using a Hybrid Metrics, Modeling and Utility Framework", of the 14th Int. Conference on Software Engineering and Knowledge Engineering, 2002, Ischia, Italy.
- [6] D. Unwin, "Introductory Spatial Analysis", Methuen & Co. Ltd. 1981, London, UK. 207p
- [7] J. Bourne, H. Liu, D. Orogio, G. Collins, N. Uckun, A. Brodersen, "Organizing and Understanding Beliefs in Advice-Giving Diagnostics System". IEEE Transaction on Knowledge and Data Engineering. 1991, pp269-280
- [8] T. Garvey "Evidential Reasoning for Geographic Evaluation for Helicopter Route Planning". IEEE Transaction on Geoscience and Remote Sensing, 1987, Vol. 25, No. 3, pp. 294-304
- [9] D. Peddle, S. Franklin, "Multisource Evidential Classification of Surface Cover and Frozen Ground", International Journal of Remote Sensing, 1992, Vol. 13, No. 17, pp.3375-338
- [10] G. Shi, "Evidential Reasoning for Geological Mapping with Multisource Spatial Data", Master Thesis, Dept. of Geomatics Engineering, The University of Calgary, Alberta, Canada, 1994.
- [11] G. Shafer, "Mathematical Theory of Evidence", Princeton University Press, 1976

Decision Support for Planning Software Evolution with Risk Management

D. Greer
Queen's University Belfast
Des.Greer@qub.ac.uk

Abstract. Software evolution planning involves a decision making process about which changes should be introduced and when. This process is informed by knowledge relating to the existing product and its environment. There will be new enhancements as well as corrections to known problems. There is also knowledge of operational risks in the system that could manifest as problems at some time in the future. To counter these, risk reduction actions may be recommended. These can be treated as candidate system changes, with their own costs and possibly extra benefits. Any system changes will also carry with them development risks. To mitigate these further risk reduction actions may be formulated, again with an associated cost. However, there is typically a limited budget for the next release of the software. This constraint, along with that of inter-dependencies between candidate changes and taking account of the stakeholders varying attitude to risk means that there is a complex decision to be made. Using the fact that systems changes are estimable in terms of cost and benefit and that risks are estimable in terms of probability and potential impact cost, an approach is described that combines software evolution with risk management, incorporating simulation and genetic algorithms to support the decision making process. The method links product and project management and provides support for those involved in planning system evolution and is evaluated here using a sample project.

1. Introduction

System evolution is an important part of the software life cycle. This evolution can be *corrective* where already logged errors are fixed, *adaptive* where a program is enhanced to meet new requirements, *perfective* where performance is improved or *preventive* where an attempt is made to correct potential faults [6]. Despite the potential disruption of software failure, preventive maintenance receives less attention. Preventative maintenance accounts for typically 5% of maintenance activities, according to one empirical study [7]. One of the reasons for this might be an apparent oversight of the need for risk management relating to the software

maintenance phase [1]. Hence, there is a need to ensure that risk management is performed as part of the software evolution process, including an assessment of *operational* risks in the existing system and *development* risks associated with any planned changes. Indeed there is an extra sensitivity to both types of risks in a maintenance situation due to the reduced opportunity for contingency planning (due to the constraint of having a legacy system) and the increased impact of risks occurring (due to a larger user community to possibly suffer) [1].

Where risks are identified and considered serious enough, risk reduction actions must be considered. For example, an operational risk could be that some part of the software may fail under a certain set of conditions. In this case a risk reduction action could be to amend the software to avoid or handle the fault. A development risk could be that the building of some component takes longer than expected. This might be mitigated by building a prototype to test the concept. In both cases there is an associated cost in making the risk reduction.

Current approaches to planning system evolution, typically involve the change manager distributing their budget among bug fixes and identified enhancements based on subjective benefit estimates and without the knowledge of the real cost of these and of potential problems that may be present as risks. However, in planning the evolution of an existing software system, the system planner must consider, alongside estimated benefits, the costs of i) enhancements; ii) corrections of known problems; iii) risk reduction for identified operational risks and iv) risk reduction for development risks associated with planned system changes.

The next section will describe the problem more formally. Section three will describe the proposed solution to the problem. Section four will describe a sample project and will illustrate the proposed evolution process. In the final section, some discussion on the results and indication of future research is provided.

2. Problem Description

i) After the review stage for an existing system there is a set of candidate changes, C. These changes have initially

been established from suggested enhancements and corrections to the existing system. Each change is described by a textual description, t , a cost value, e and a benefit value, b .

$$C = \{(t_1, e_1, b_1), (t_2, e_2, b_2), \dots (t_n, e_n, b_n)\}$$

ii) Also in the existing system, a set of operational risks O can be identified. These are represented by triplet of user-assigned values consisting of a description o_i , probability value, p_i and impact value i_i .

$$O = \{(o_1, p_1, i_1), (o_2, p_2, i_2), \dots (o_n, p_n, i_n)\}, \text{ where } 0 \leq p_i \leq 1$$

iii) Considering this set of risks may reveal the need for risk reduction actions, A , which may also be treated as changes. Thus, the set of changes is extended to C'

$$C' = A \cup C$$

(iii) Each change in C' can alter the probability or impact values of any of the identified operational risks in O . Thus, we define a function Φ_C

$\Phi_C: (o_i, p_i, i_i) \rightarrow (o_i, p'_i, i'_i)$ where p'_i is new probability after change C and i'_i is the new impact value after change C .

(v) Changes may also introduce new risks. Thus for change C_i there is a new set of risks, X . Should that change occur the set of operational risks is expanded to O' where $O' = O \cup X$

(vi) Each change in C' may have associated with it a set of development Risks, D . This set consists of the quadruplet: the identity of the change c triggering the risk, the description of the risk, d , the probability, p of the risk occurring and the impact i of the risk occurring.

$$D = \{(c_1, d_1, p_1, i_1), (c_2, d_2, p_2, i_2), \dots (c_n, d_n, p_n, i_n)\}, \text{ where } c_i \in C, 0 \leq p_i \leq 1$$

(vii) Further changes, M can be defined to reduce development risks. These contain the identity of the development risk d , a description t , a cost e and an additional benefit b .

$$M = \{(d_1, t_1, e_1, b_1), (d_1, t_2, e_2, b_2), \dots (d_1, t_n, e_n, b_n)\} \text{ where } d_i \in D$$

Thus, $C'' = M \cup C'$, $D' = \{(c_1, d_1, p'_1, i'_1), (c_2, d_2, p'_2, i'_2), \dots (c_n, d_n, p'_n, i'_n)\}$

(viii) We will eventually choose a subset C^* from C'' , but there may be cases of mutual exclusion. Therefore, we define a binary relation μ on C'' such that $(x_i, x_j) \in \mu$ implies $((x_i \in C^*) \wedge (x_j \in C^*)) = \text{FALSE}$

(ix) There may also be cases where candidate changes should be selected together or not at all. Therefore, we define a binary relation ν on C'' such that $(x_i, x_j) \in \nu$ implies $((x_i \in C^*) \text{ XOR } (x_j \in C^*)) = \text{FALSE}$

x) Suppose a function, χ returns the mean cost of a set of changes and a function, β returns the mean benefit accrued from those changes and that there is a user-set limit of L on the mean cost, then a set C^* can be selected from C'' .

$$C^* \subseteq C'' : \chi(C^*) < L, [\chi(C^*) - \beta(C^*)] \text{ is maximised}$$

xi) In any product there may be an operational risk exposure level that is acceptable. This is represented by the probability that the mean impact cost (ι) from a simulation using the remaining operational risks in O' is equal to or exceeds the acceptable probability, p , that a set impact cost (γ) is incurred.

$$P(\iota \leq \gamma) \leq p$$

xii) There may also be a development risk exposure level that is acceptable. This is represented by the probability that the mean impact cost (ϕ) from a simulation using the remaining development risks in D' is equal to or exceeds the acceptable probability, q , that a set impact cost (κ) is incurred. $P(\phi \leq \kappa) \leq q$

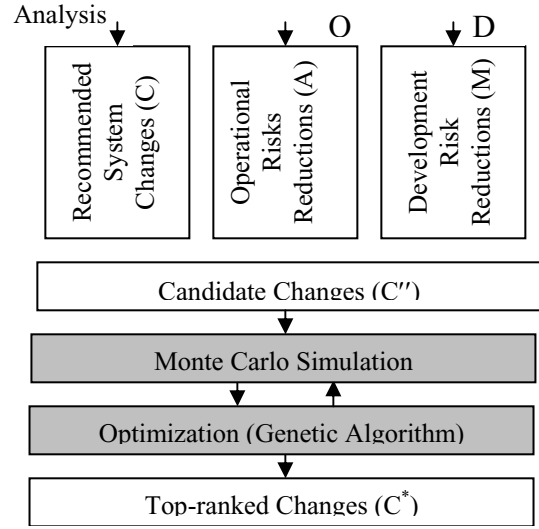


Figure 1: Decision Support Process

3. Decision Support Solution

Figure 1 describes the approach taken to supporting system evolution decision support. The process starts with an analysis of the current system. This can be by traditional methods, although in previous work, a goal oriented approach where the existing system is compared against an ideal model of the system, has proved to be particularly effective at revealing shortcomings and also suitable for identifying operational risks in the current product [4]. This analysis reveals required system changes (C) and operational risks (O). Consideration of these risks can induce further system changes that will reduce these risks (A). These are added to the system changes to produce a new set of candidate changes (C'). These in turn, are considered for possible development risk (D). Here it is possible to use risk questionnaires such as those from the SEI [3]. From these, further risk reductions may be established (M) and added to form an updated set of candidate changes (C''). The probabilities for the risks are used to create a discrete probability distribution of costs

using Monte Carlo simulation. At the end of each Monte Carlo simulation, the genetic algorithm executes. Together the simulation and the genetic algorithm select changes from the candidate changes to form a number of solutions (C^*).

Genetic algorithms have been shown to be appropriate in situations where there are complex relationships between several factors and consequently a very large solution space [2] and indeed have already been successfully employed in similar problems [5].

4. Case Study

In order to illustrate the method, a sample system will be used, where 10 operational risks have been identified by risk analysts. Table 1 provides the data for these, the risks being labelled OR_n and each risk having with it a probability of occurrence over the lifetime of the system and an estimated impact in cash values.

Table 1: Operational risks for existing system in sample project

Operational Risk	Probability	Impact (000's)
OR_1	0.1	8
OR_2	0.5	7
OR_3	0.2	8
OR_4	0.4	14
OR_5	0.6	16
OR_6	0.7	11.5
OR_7	0.9	11
OR_8	0.1	10.5
OR_9	0.4	17.5
OR_{10}	0.3	6.5

As a result of analysis, considering existing problems and opportunities for innovation a number of system changes have been identified as shown in Table 2. Further, consideration of the operational risks induces additional risk reduction changes, as shown in table 3. These have the effect of reducing the risk exposure due to the risk by either reducing the probability of the risk or the impact of the risk, or both. In Table 3, C_{11} – C_{15} have been added to reduce risks OR_5 – OR_9 . In most cases there is no direct benefit from the change, but to illustrate the flexibility of our approach, one change (C_{14}) accrues a small benefit if implemented. In addition, table 4 shows that some of the risks are addressed by already defined by system changes. To mimic the real-world possibilities, in some cases a combination of changes is required to achieve the risk reduction (e.g. C_1+C_2 to reduce OR_1).

Table 2: System Changes with Costs and Benefits

Change	Cost (\$000's)	Benefit (\$000's)
C_1	16	56
C_2	17	54
C_3	16	102
C_4	19	72
C_5	15	86
C_6	8	67
C_7	23	97
C_8	10	36
C_9	17	46
C_{10}	14	94

Table 3: Operational Risk Reductions

Change	Cost (000's)	Benefit (000's)	Risk Reduced	New Prob-ability	New Impact (000's)
C_{11}	8	0	OR_5	0	0
C_{12}	7	0	OR_6	0.1	5
C_{13}	6.5	0	OR_7	0.2	2
C_{14}	9	4	OR_8	0	0
C_{15}	5.5	0	OR_9	0	0

Table 4: Risk reductions via system changes

Change	Risk Reduced	New Probability	New Impact (000's)
C_1+C_2	OR_1	0	0
C_2	OR_1	0.1	3
C_6+C_7	OR_{10}	0	0
C_{14}	OR_{10}	0.2	4
C_2	OR_2	0.4	2
C_2	OR_3	0.1	4
C_3	OR_4	0.1	14

While clearly not a desirable effect, the introduction of new operational risks as a result of the some system changes may occur. This is illustrated in Table 5.

Table 5: Operational Risks Introduced by Changes

Risk	Change	Probability	Impact (000's)
OR_{11}	C_{15}	0.3	5
OR_{12}	C_2	0.4	4

A further factor is that system changes may have risks associated with the development process. Typically these are risks relating to budget or schedule and may include personnel risks, technical uncertainties, changing requirements, and so on. These development risks for the sample project are illustrated in Table 6.

For each development risk, there is also the possibility of one or more risk reduction actions. These potentially

reduce the probability and/or impact of the risk (Table 7) but have a cost associated with them.

To represent inherent dependencies, pairs of system changes were deemed to be mutually exclusive:

$$\mu = \{(C2, C3), (C4, C6)\} \quad (1)$$

Similarly, the following pair was deemed to be coupled:

$$v = \{(C5, C7), (C7, C10)\} \quad (2)$$

Table 6: Development Risk Assessment

Development Risk	Change	Probability	Impact (000's)
DR ₁	C ₂	0.3	5
DR ₂	C ₂	0.5	5
DR ₃	C ₄	0.4	16
DR ₄	C ₄	0.5	5
DR ₅	C ₆	0.6	2
DR ₆	C ₆	0.8	5
DR ₇	C ₇	0.5	10
DR ₈	C ₈	0.9	4
DR ₉	C ₈	0.7	3
DR ₁₀	C ₉	0.2	10
DR ₁₁	C ₁₀	0.1	4
DR ₁₂	C ₁₀	0.2	10

The overall mean budget for the sample project was set at 130K, so that no solution was allowed that exceeded this. A further constraint was in the levels of risk exposure in the proposed system and in the development process. For operational risks, the constraint was set that there should not be greater than a 20% chance of a 10K impact. For development risk, a 10% chance of a 10K overspend was acceptable.

Table 7: Risk Reduction Actions on Development Risks

Development Risk	Risk Reduction	Cost (000's)	New Probability	New Impact (000's)
DR ₁	DRR ₁	1	1	5
DR ₂	DRR ₂	2	2	2
DR ₂	DRR ₃	2	1	1
DR ₃	DRR ₄	2	0	0
DR ₄	DRR ₅	2	0	0
DR ₅	DRR ₆	1	3	2
DR ₅	DRR ₇	1	0	0
DR ₆	DRR ₈	2	0	0
DR ₇	DRR ₉	1	1	6
DR ₈	DRR ₉	1	0	0
DR ₉	DRR ₉	1	1	3
DR ₁₀	DRR ₁₀	3	2	4
DR ₁₁	DRR ₁₀	3	1	2
DR ₁₂	DRR ₁₀	3	1	8

4.1. Simulation

Simulation was achieved by Monte Carlo Analysis with Latin Hypercube sampling. This is made possible due to the fact that we have a probability value and an impact value for each of the risks identified, allowing a discrete probability distribution to be generated. The probability value is assumed to be over the lifetime of the system. In the sample project an arbitrary 1000 iterations were used as a compromise between obtaining a good distribution and execution time. When applying Monte Carlo simulation to existing operational risks (Table 1 and 4), the probability value is that achieved after any risk reducing change has occurred if that change has been selected. Also operational risks introduced into the system (Table 5), will only apply if a candidate system change has been selected for implementation. Development risks (Table 6) are also represented by a simulated probability distribution based on the estimated probability and impact of each risk, if it occurs. These are also only taken into account if the associated change has been selected. Changes are selected using a genetic algorithm. This and the Monte Carlo analysis were performed using the RiskOptimizer software tool from Palisade [8].

Each system change (table 2) along with each risk reduction activity (table 3) is considered a candidate action. The genetic algorithm is used to select which of these are taken. The crossover routine used is that of uniform crossover as described in [2] and recommended in [9]. This involves randomly choosing items in each of 2 selected parents to use to create an offspring. The percentage in the first parent chosen is determined by the crossover rate. The decision to include a change or not is a binary one, so that crossover means taking some of the changes from one chromosome (generated solution), adding these to a new chromosome and then adding the changes from a different selected chromosome. Mutation is performed on the offspring by generating a random number between 0 and 1 for each item. If the number is less than or equal to the mutation rate then that variable is mutated. In our case the decision is whether or not a system change is included. In this case a mutation is effectively moving from exclusion to inclusion or vice versa. Following preliminary investigations a crossover rate of 0.5 was used and an auto-mutation routine employed that adjusted the mutation rate automatically when a population ceased to improve.

4.2. Sample Results

Table 8 provides details of the results produced using the data from the case study. The objective was to maximise the profit from the activities selected.

The results show a general exclusion of changes C₁, C₃ and C₆ from the enhancement type changes and of C13

from the operational risk reductions. This is probably a result of the mutual exclusion set μ (1). In the sample project, development risk reduction DRR₅ is included in 7 of the top ten solutions with occasional inclusion of DRR₆, DRR₇ and DRR₈ in the top ten. C₁, C₃ and C₆ did occur much further down in the rankings.

In practice, an analyst may take the top-ranked solution but also consider the third ranked solution which, in this case, has the same content but includes the development risk reductions of DRR₅ and DRR₆ with a loss in profit of 1K. This is as an acknowledgement of the uncertainty of the input data and that the analyst should be able to take account of their own judgement.

Note that the profit values also include any risks that were realised in the simulation. The operational risk reduction, C₁₃ and several development risk reductions did not get selected in any of the top ten solutions, implying that they were not worth the effort within the limitations of the project.

Table 8: Top ten ranked solutions

	Mean Profit	Min.	Max.	Rank	
	129.6	86.5	172.5	1	
	129.1	83.5	169.5	2	
	128.6	85.5	171.5	3	
	127.6	84.5	170.5	4	
	117.1	63.5	167.0	5	
	116.6	60.5	164.0	6	
	116.1	62.5	166.0	7	
	115.6	59.5	163.0	8	
	115.1	68.0	158.5	9	
	112.0	62.5	154.5	10	
C ₂	✓	✓	✓	✓	✓
C ₄	✓	✓	✓	✓	✓
C ₅	✓	✓	✓	✓	✓
C ₇	✓	✓	✓	✓	✓
C ₈	✓	✓	✓	✓	✓
C ₉	✓	✓	✓	✓	✓
C ₁₀	✓	✓	✓	✓	✓
C ₁₁	✓	✓	✓	✓	✓
C ₁₂	✓	✓	✓	✓	✓
C ₁₄	✓	✓	✓	✓	✓
C ₁₅	✓	✓	✓	✓	✓
DRR ₅	✓	✓	✓	✓	✓
DRR ₆	✓	✓	✓	✓	✓
DRR ₇	✓	✓	✓	✓	✓
DRR ₈	✓	✓	✓	✓	✓

5. Conclusions and Future Research

An approach for supporting the decision process in planning the evolutions of existing systems has been described. The approach is novel in 3 ways. (i) As well as

considering corrective and enhance work, operational risks in the current system are considered as identification sources for system change. (ii) Risks are modelled along with predicted costs using estimated probability and impact values and using Monte Carlo simulation. This is not novel in financial applications, but is regarding software evolution. (iii) A genetic algorithm has been used to optimise the system evolution plan for profit but maintaining a cost limit, taking account of inherent dependencies and ensuring development and operational risk exposure levels are kept below some limit. Overall, this decision support approach offers an improvement on the ad hoc approach currently prevalent, in taking account of risks as well as costs and benefits allows for a truer prediction of the likely costs of selecting certain changes, of accepting certain risks and mitigating others.

There remains some practical problems in collecting cost, benefit and risk data. One promising approach, from a previous industrial study [4] is the use of symbols to represent costs, benefits and probabilities and impact costs for risks. This approach involved choosing phrases such as “high” to represent the measure and mapping this to a preset value. This work was successful in capturing cost and risk data, requiring minimal expertise from the project planner, other than past experience. Incorporating this work further constitutes future work which will also involve further empirical studies of the approach, and the various trade-offs involved.

6. References

- [1] Charette, R.N.; Adams, K.M.; White, M.B, Managing Risk in Software Maintenance, *IEEE Software*, Volume: 14 (3), 1997. pp. 43 – 50.
- [2] De Jong, K.A. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Doctoral Thesis, Univ. Michigan, Ann Arbor 1975.
- [3] Dorofee, A., Walker, J., Alberts, C.J., Higuera, R.P., Murphy, R.L. and Williams, R.C., *Continuous Risk Management Guidebook*, Software Engineering Institute, Carnegie Mellon University, 1996.
- [4] Greer, D., Bustard, D. and Sunazuka, T. Prioritisation of System Changes using Cost-Benefit and Risk Assessments, Fourth IEEE International Symposium on Requirements Engineering, pp 180-187, June, 1999.
- [5] Greer, D. & Ruhe, G., Software Release Planning: An Evolutionary and Iterative Approach, *J. Information and Software Technology*, vol. 46, issue 4, pp 243-253, 2004.
- [6] IEEE. 1993. *IEEE Standard for Software Maintenance*. IEEE Std 1219-1993. IEEE, New York, NY.
- [7] Kemerer, C.F. & S. Slaughter, Determinants of Software Maintenance Profiles: An Empirical Investigation, *J. Software Maintenance*, vol. 9, 1997, pp. 235-251.
- [8] Palisade Corporation, *Guide to RISKOptimizer: Release 1.0*, October, 2001, Palisade Corp.
- [9] Spears, W.M.; De Jong, K.A., On the virtues of parameterized uniform crossover, *Proc. International Conference on Genetic Algorithms*, pp. 230-237, 1991.

Machine Learning-Based Quality Predictive Models: Towards an Artificial Intelligence Decision Making System

Hakim Lounis

*Department of Computer Science
Université du Québec à Montréal,
Canada
lounis.hakim@uqam.ca*

Lynda Ait-Mahiedine

*Department of Computer Science and
Operations Research
Université de Montréal, Canada
aitmemel@iro.umontreal.ca*

Abstract. The ISO/IEC international standard (14598) on software product quality states that “Internal metrics are of little value unless there is evidence that they are related to external quality”. Many different approaches have been proposed to build such empirical assessment models. Different Machine Learning (ML) algorithms are explored with regard to their capacities of producing predictive models. The predictability of each model is then evaluated and their applicability in an Artificial Intelligence (AI) decision-making system is discussed.

1 Introduction

Integration of metrics computation in most popular CASE tools is a marked tendency. Software metrics provide quantitative means to control the software development and the quality of software products. They are necessary to identify where the resources are needed, and are a crucial source of information for decision-making. Moreover, early availability of metrics is a key factor to a successful management of software development.

As it is stated by the ISO/IEC international standard (14598), internal metrics are especially helpful when they are related to external quality attributes, e.g., maintainability, reusability, etc. Predictive models can take different forms depending on the building technique that is used. For example, they can be mathematical models (case of statistical techniques like linear and logistic regression) or AI-based models (case of ML techniques). In all cases, they allow affecting a value to a quality characteristic based on the values of a set of software measures, and they allow the detection of design and implementation anomalies early in the software life cycle. They also allow organizations that purchase software to better evaluate and compare the offers they receive.

As far as we know, Selby and Porter [1] have been the first to use a ML classification algorithm to automatically construct software quality models. They

have used ID3 [2], a ML classification algorithm, to identify those product measures that are the best predictors of interface errors likely to be encountered during maintenance. After Selby & Porter, many others, e.g., [3], [4], have used ML classification algorithms to construct software quality predictive models. More recently, De Almeida & al. [5] have investigated ML algorithms with regard to their capabilities to accurately assess the correctness of faulty software components. Three different families algorithms have been analyzed on the same data than those used by [3], and FOIL [6], an inductive logic programming system, presented the best results from the point of view of model accuracy.

In the present work we reaffirm that machine-learning approaches, as they are classified in a well-accepted taxonomy [8], are of a significant help for the building of software quality predictive models. This statement is strengthened when the main purpose is the conception of a Knowledge-Based System (KBS) for quality assessment. KBSs are used in numerous application domains. They are used to reproduce an expert's reasoning and are based on two distinct components: knowledge and reasoning. Separation between these two levels of intervention makes it possible to offer a flexibility of operation that many traditional software approaches are missing. KBSs are presently an effective and useful solution to integrate the necessary analyses of software engineer experts and to meet the needs of the software industry, in terms of quality. It is especially true when we consider the recent progress done within the tools that help KBSs design.

In the balance of this paper, we first briefly present in section 2 the different ML approaches we have analyzed. In section 3, we introduce the three working hypotheses we have stated and verified with the ML algorithms. Section 4 provides the experimental results we have obtained. Finally, conclusions, current research and directions for future research are outlined and the architecture of a KBS for quality assessment is proposed.

2 Artificial Intelligence Approaches

Machine Learning is a prolific subfield of AI where many approaches have been developed. The taxonomy [8] in figure 1 illustrates only a subset of them:

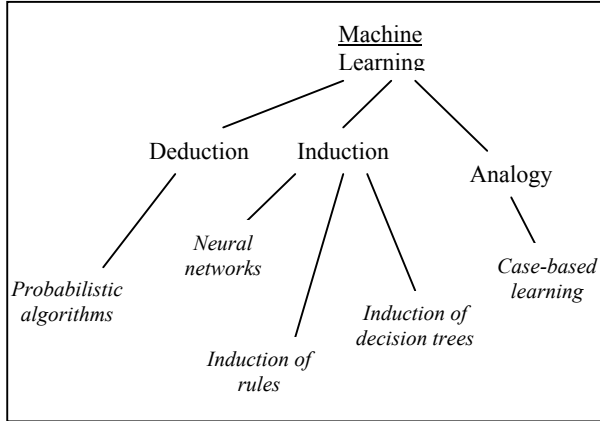


Figure 1. Machine Learning Taxonomy

Most of the work done in machine learning has focused on supervised machine learning algorithms. Starting from the description of classified examples, these algorithms produce definitions for each class. The most popular approach is an induction one - the divide and conquer approach -. In this approach, a *decision tree* generally represents the induced knowledge. It is the case of algorithms like ID3 [2], CART [9], ASSISTANT [10] and C4.5 [11]. This algorithm could summarize their principle:

```

If all the examples are of the same class
Then- Create a leaf labeled by the class name;
Else
- Select a test based on one attribute;
- Divide the training set into subsets, each
  associated to one of the possible values of the
  tested attribute;
- Apply the same procedure to each subset;
Endif.
  
```

The key step of the algorithm above is the selection of the “best” attribute to obtain compact trees with high predictive accuracy. Information-based heuristics have provided effective guidance for the division process.

OC1 (Oblique Classifier 1) is also a decision tree induction system designed for applications where the instances have numeric (continuous) feature values, which is the case in our study. However, it builds decision trees that contain linear combinations of one or more attributes at each internal node:

$$\sum_{i=1}^{i=k} a_i x_i + a_{k+1}$$

These trees then partition the space of examples with both oblique and axis-parallel hyper planes. More details on this algorithm are given in [12].

The *induction of rules* is another important ML branch. Because the structure underlying many real-world datasets is quite rudimentary, and just one attribute is sufficient to determine the class of an instance quite accurately, it turns out that simple rules frequently achieve surprisingly high accuracy. The pseudo-code for such an approach (called one rule) is the following:

```

For each attribute
  For each value of that attribute, make a rule as
  follows:
  - Count how often each class appears;
  - Find the most frequent class;
  - Make the rule assign that class to this
    attribute value;
Calculate the error rate of the rules;
Choose the rules with the smallest error rate.
  
```

The covering algorithms illustrate a more sophisticated approach. They represent classification knowledge as a disjunctive logical expression defining each class. CN2 [13] is an instance of such a family. The following algorithm could summarize it:

```

- Find a conjunction that is satisfied by some
  examples of the target class, but no examples from
  another class;
- Append this conjunction as one disjunct of the
  logical expression being developed;
- Remove all examples that satisfy this conjunction
  and, if there are still some remaining examples of
  the target class, repeat the process.
  
```

In the *case-based learning* (CBL) approach the training examples are stored verbatim, and a distance function is used to determine which member(s) of the training set is closest to an unknown test instance. Once the nearest training instance(s) has/have been located, its class is predicted for the test instance. Although there are multiple choices, most instance-based learners use Euclidian distance. Despite several practical problems, it is considered as a good compromise.

The multilayer perceptron is probably the most widely used *neural network* architecture to solve classification problems with supervised learning. It consists of an input layer, one or more hidden layers, and an output layer of neurons that feed one another via synaptic weights. The input layer simply holds the data to be classified. The outputs of the hidden and output layers are computed, for each neuron, by calculating the sum of its inputs multiplied by its synaptic weights and by passing the result through an output function. The

synaptic weights of the hidden and output neurons are determined by a training procedure where examples of the patterns to be learned are presented successively, and where the weights are adjusted so as to minimize the error between the obtained classification results and the desired ones. The standard training procedure for the multilayer perceptron uses the backpropagation algorithm [14] or one of its derivatives. For instance, the resilient backpropagation (RPROP) algorithm [15] has faster learning and a better overall performance than regular backpropagation or backpropagation with a momentum [14].

Lastly, Bayesian techniques have long been used in the field of pattern recognition, but only recently have they been taken seriously by ML researchers [16] and made to work on data sets with redundant attributes and numeric attributes. Such a *probabilistic* algorithm is trained by estimating the conditional probability distributions of each attribute, given a class label. The classification of a case, represented by a set of values for each attribute, is accomplished by computing the posterior probability of each class label, given the attributes values, using Bayes' theorem. The case is assigned to the class with the highest posterior probability. The following formula corresponds to the probability of the class value $C=c_j$ given the set of attribute values $e_k=\{A_1=a_{1k}, \dots, A_m=a_{mk}\}$:

$$p(c_j / e_k) = \frac{\prod_{k=1}^m p(a_{ik} / c_j) p(c_j)}{\sum_{h=1}^c \prod_{k=1}^m p(a_{ik} / c_h) p(c_h)}$$

Recent empirical evaluations have found the Bayesian algorithms to be accurate [17] and very efficient at handling large databases (e.g., data mining tasks). The simplifying assumptions underpinning the Bayesian algorithms are that the classes are mutually exclusive and exhaustive and that the attributes are conditionally independent once the class is known. It is a great stumbling block; however, some attempts are being made to apply Bayesian analysis without assuming independence.

3 Working Hypotheses

The goal of this work is double: (i) to prove that ML algorithms can have predictive accuracies as good as those of other approaches, and, (ii) to consider the use of the learned models within a knowledge-based architecture. To do that, we consider hypotheses linking internal measures to three different quality attributes.

The first hypothesis (let us call it, H_1) concerns the relevance of some internal product measures for assessing corrective maintenance costs. Many different approaches have been proposed to build corrective

maintenance estimation/evaluation models. To do so, and as in [3] and [5], we have used (1) data collected on corrective maintenance activities for the Generalized Support Software reuse asset library located at the Flight Dynamics Division of NASA's GSFC and (2) internal product measures extracted directly from the faulty components of this library, e.g., size metrics, cyclomatic complexity, Hasteed's metrics, etc. The corrective maintenance data come from the maintenance of a library of reusable components. We have dichotomized the corrective maintenance cost into two categories: low and high. For more details about the data, please see [5].

The second hypothesis is about the impact of three internal characteristics (inheritance, coupling and complexity) of OO applications on reusability. Reusability is a complex factor, which is domain dependent. Some components are more reusable in one domain than in others. Our goal is not to search for a set of methods measuring reusability universally but to study some specific aspects and characteristics pertaining to OO programming languages, e.g. C++ that affect reusability. We propose four reusability hypotheses (H_{21} , H_{22} , H_{23} , and H_{24}) regarding the relationships between reuse and each of inheritance, coupling (at the code and design level), and complexity, respectively. Different aspects can be considered to measure empirically the reusability of a component depending on the adopted point of view. One aspect is the amount of work needed to reuse a component from a version of a system to another version of the same system. Another aspect is the amount of work needed to reuse a component from a system to another system of the same domain. This latter aspect was adopted as the empirical reusability measure for our experiments. To define the possible values for this measure, we worked with a team specializing in developing intelligent multiagents systems. The obtained values are:

1. Totally reusable: means that the component is generic to a certain domain (in our case "intelligent multiagents systems").
2. Reusable with minimum rework: means that less than 25% of the code needs to be altered to reuse the component in a new system of the same domain.
3. Reusable with high amount of rework: means that more than 25% of the code needs to be changed before reusing the component in a new system of the same domain.
4. Not reusable at all: means that the component is too specific to the system to be reused.

For more information about the data, see [7].

Finally, the goal of the third hypothesis is to empirically investigate the relationships between object-oriented design measures and fault-proneness at the class level.

We select a practical measure of fault-proneness as the dependent variable for our study: in a non-faulty component, there was not any change of corrective type and in a faulty one, there were one or more changes of corrective type during the development/maintenance phase. The measures of inheritance, coupling, and cohesion identified in a literature survey on object-oriented design measures are the independent variables used in this study; we obtain 3 more hypotheses called H_{31} , H_{32} , and H_{33} . We focus on design measurement since we want the measurement-based models investigated in this study to be usable at early stages of software development. Furthermore, we only use measures defined at the class level since this is also the granularity at which the fault data could realistically be collected. More details about used data are in [18].

4 Hypotheses Verification and Predictive Models Accuracy Computation

Figure 2 illustrates the different steps of the empirical process we follow to verify the 8 hypotheses stated in section 3.

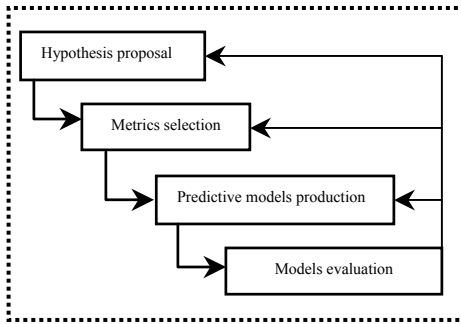


Figure 2. Empirical Process

From the ML taxonomy presented in section 2, we select 7 algorithms representing different approaches. Finally, the computation of models accuracy is done thanks to a cross-validation procedure. It is helpful when the amount of data for training and testing is limited; we try a fixed number of approximately equal partitions of the data, and each in turn is used for testing while the remainder is used for training. In the end, every instance has been used exactly once for testing.

Table 1 summarizes the accuracy computed for each of the 8 models. Dark grey cells show accuracies greater than 75%, and light grey ones contain accuracies between 60% and 75%.

The accuracies computed for yet studied ML algorithms are confirmed by this study. As we were expecting, C4.5 (decision tree), CN2 (covering rules), and the RPROP neural network present the best result in terms of

predictive power. Even, simplistic algorithms (e.g., one rule and naïve Bayes) show some results higher than 60%! On the other hand, a basic CBL algorithm shows promising results; these results could be improved by changing, for instance, the distance function. Lastly, OC1 gives some interesting results, mainly for hypotheses H_{31} , H_{32} , and H_{33} .

Table 1. Computed Accuracies (%)

Stated Hypotheses	H_1	H_{21}	H_{22}	H_{23}	H_{24}	H_{31}	H_{32}	H_{33}
ML algos								
C4.5	66	73.8	86.9	88.1	89.3	73.8	77.5	73.7
OC1	68.8	48.8	51.2	53.6	54.8	72.5	73.8	70.9
One-rule	58.5	45.2	53.6	56	42.9	72.5	76.3	65.8
CN2	54	67.3	62.5	63.1	60	87.5	78.5	72.2
CBL	56.7	54.8	66.7	63.1	60.7	73.8	71.3	70.9
RPROP neural net	56.7	56.3	75	75	68.8	81.3	75	81.3
Naïve Bayes	54.9	45.2	46.4	57.1	45.2	36.2	68.8	70.9

In terms of selected internal metrics, we found a certain uniformity between the different models, and we confirm some results previously published. For instance, various ML algorithms select CSB (size of the object in bytes) as a complexity metric having an impact on the reusability of the component. It is also the case for design export coupling metrics like OCAEC and OMMEC. They are selected by numerous ML algorithms as relevant for assessing the reusability of a C++ class. On the other hand, the classic CBO metric (the number of other classes to which a class is coupled), and methods invocation, e.g., ACMIC (ancestor class method import coupling), RFC_{OO} (the number of methods that can potentially be executed in response to a message received by an object of that class), and IH-ICP (the number of ancestor methods invocation in a class, weighted by the number of parameters of the invoked methods) are identified as relevant design coupling measures for fault-proneness assessment. In the case of inheritance measures, NMI (the number of methods inherited), NMO (the number of overridden methods), and DIT (the maximum length from the class to a root) are the three that are retained as relevant for our fault-proneness hypothesis.

5 Towards an AI Decision Making System

This work aims first at showing that ML algorithms are a serious alternative for the production of quality predictive models. Despite the fact that obtained accuracies are broadly high, we know that for some ML algorithms, e.g., C4.5 and CN2, the results will be much higher with non-continuous numeric data. Our past work

on the fuzzyfication of such measures [7] is an example of such a data pre-processing step. It also gives a solution to the problematic use of precise metric thresholds values. We are now working on a pre-processing tending to discretize such attributes, based on statistical considerations.

On the other hand, the main strength of ML produced models is that we can incorporate them in an AI decision-making process, where, a KBS architecture keeps a good separation between what we consider as an expert knowledge (the produced models) and the procedures that exploit this knowledge. We are working on an object-oriented (OO) knowledge-based architecture, which allows us (i) analyze OO source code, (ii) compute internal metrics, (iii) produce ML-based predictive models, and (iv) exploit these models by an inference system. Even the KBS is object-oriented, as we use the ILOG Jrules¹ API for build it. Figure 3 illustrates such an architecture.

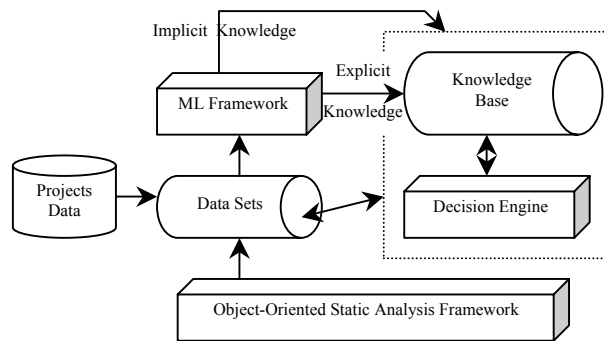


Figure 3. An OO KBS for quality assessment

One distinction between C4.5 and CN2 (for instance) from one part, and RPROP from the other part is on the induced knowledge. The two first produce explicit pieces of knowledge, e.g., rules or decision trees, that could be stored in a knowledge base and exploited by a knowledge-based system in a decision making process. RPROP doesn't produce any kind of explicit knowledge. However, it allows predicting a quality attribute value, giving some internal measures values. In this case, we talk about implicit knowledge.

References

- [1] A. Porter, R. Selby, "Empirically-guided software development using metric-based classification trees", IEEE Software, 7(2):46-54, 1990.
- [2] J.R. Quinlan, "Induction of decision tree", Machine Learning journal 1, p 81-106, 1986.

- [3] V. Basili, S. Condon, K. El Emam, R. B. Hendrick, W. L. Melo, "Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components", In Proc. of the IEEE 19th Int'l. Conf. on S/W Eng., Boston, MA, 1997.

- [4] M. Jorgensen, "Experience with the Accuracy of Software Maintenance Task Effort Prediction Models", In IEEE TSE, 21(8):674-681, 1995.

- [5] M.A. De Almeida, H. Lounis, W. Melo, "An Investigation on the Use of ML Models for Estimating Software Correctability", In the Int. Journal of Software Eng. and Knowledge Eng., special issue on Knowledge Discovery from Empirical Software Engineering Data, 1999.

- [6] J.R. Quinlan, "Learning Logical Definitions from Relations", In ML journal, vol 5, n°3, p 239-266, 1990.

- [7] H.A. Sahraoui, M. Boukadoum, H. Lounis, "Building Quality Estimation models with Fuzzy Threshold Values", In "L'objet", volume 7, numéro 4, 2001.

- [8] Y. Kodratoff, "Apprentissage symbolique : une approche de l'IA", tome 1&2, Cépaduès-Editions, 1994.

- [9] L. Breiman, J. Friedman, R. Olshen, C. Stone, "Classification and Regression Trees", Published by Wadsworth, 1984.

- [10] B. Cestnik, I. Bratko, I. Kononenko, "ASSISTANT 86: a knowledge elicitation tool for sophisticated users", Progress in machine learning, Sigma Press, 1987.

- [11] J.R. Quinlan, "C4.5: Programs for Machine Learning", Morgan Kaufmann Publishers, 1993.

- [12] S.K. Murthy, S. Kasif, S. Salzberg, "A System for Induction of Oblique Decision Trees", Journal of Artificial Intelligence Research 2, 1-32, 1994.

- [13] P. Clark, T. Niblett, "The CN2 induction algorithm", In ML Journal, 3(4):261-283, 1989.

- [14] D. E. Rumelhart, J. L. McClelland, ed., Parallel Distributed Processing, Explorations in the Microstructure of Cognition; vol. 1: Foundations, MIT press, 1986.

- [15] M. Riedmiller, H. Braun, "A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm", Proc. of the IEEE Intl. Conf. on Neural Networks, San Francisco, CA, 1993.

- [16] P. Langley, W. Iba, K. Thompson, "An analysis of Bayesian Classifiers", In Proc. of the National Conference on Artificial Intelligence, 1992.

- [17] R. Kohavi, "Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid", In Proc. of the 2nd Int. Conference on Knowledge Discovery and Data Mining, 1996.

- [18] L.C. Briand, J. Wust, H. Lounis, "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs", In Empirical Soft. Engineering, an Int. Journal, 6 (1):11-58, 2001, Kluwer Academic Publishers.

¹ ILOG: www.ilog.com

Requirements for a Tool in Support of SE Technology Selection

Andreas Jedlitschka and Dietmar Pfahl

Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, 67661 Kaiserslautern, Germany
{Andreas.Jedlitschka; Dietmar.Pfahl}@iese.fraunhofer.de

Abstract. *Decision support in software engineering is an emerging field. The need to select the best method, technique or tool in a given business context is becoming increasingly important. In today's software development organizations, technologies are employed that frequently lack sufficient evidence regarding their suitability, their limits, qualities, costs, and inherent risks. This paper presents ongoing research towards the development of a decision support system that aims at improving software engineering technology selection by software managers. To develop such a system, a multiple-step requirements analysis, consisting of a literature survey, a pilot study amongst research managers, and the analysis of additional use cases, was performed. The focus of this paper is on presenting and discussing the results of this three-step requirements analysis process.*

1. Introduction

Software engineering decision support (SE-DS) is an emerging field [1, 2]. One of the major goals of SE-DS is to support software managers in selecting suitable SE technologies. Suitability implies the existence of a defined level of evidence about the effectiveness of a specific SE technology in a given context. Similar to work done in the area of empirical software engineering (ESE), SE-DS implies data collection, analysis, and modeling. In addition, SE-DS involves model application, possibly supported by software infrastructure.

In this paper, we rely on a new approach to comprehensive SE-DS [3]. This approach does not require process simulators but enhances the power of existing software engineering decision support systems (SE-DSS) that focus on providing local evidence. Examples of such systems include the ESERNET repository [4, 5]. The trade-off as compared to using process simulators is that technology interaction and development context can only be modeled in a black box like manner via pre/post-conditions.

The focus of this paper is on requirements elicitation for comprehensive SE-DSS serving software managers in

making SE technology selection decisions aligned to project goals and/or business objectives.

The structure of the paper is as follows. Section 2 reviews the sets of generic requirements to SE-DSS proposed by others. Section 3 and 4 present our requirements elicitation process for a comprehensive SE-DSS. Section 5 maps our findings to a standard architecture and a generic requirements classification framework. The paper concludes with a brief discussion of the results and an outlook to future work.

2. Generic Requirements for SE-DSS

Several authors have proposed sets of requirements for SE-DSS on various levels of abstraction. Ruhe [1], for example, suggests nine categories of SE-DSS requirements. The focus of his analysis is on requirements “that combine the intellectual resources of individuals and organizations with the capabilities of the computer to improve effectiveness, efficiency and transparency of decision-making”.

Also related to our proposal of building a comprehensive SE-DSS is the work done by Biffl et al. [6]. They describe functional and non-functional requirements of a knowledge management system that builds upon a framework to support software inspection planning.

3. Requirements Elicitation via expert interviews

The purpose of conducting expert interviews was to elicit a relevant and reliable set of requirements for a comprehensive SE-DSS. In order to be relevant, interviewees had to be sufficiently mature with regards to software management experience. In order to be reliable, a sufficient number of subjects had to be interviewed. Being a research institute that is largely involved in conducting research and transfer projects with software industry, Fraunhofer IESE offered enough experts to conduct a pilot study. In total, seven business area managers, one institute director, and one department head participated in the pilot study. Business area managers are

senior consultants who establish and maintain contacts with industrial partners, acquire projects, and help transfer research results into industrial environments. Personal industrial project experience within the group of interviewees ranged from 5 to 17 years.

We used structured interviews for requirements elicitation. Each interviewee had to answer seven questions. All questions were formulated as open questions (i.e., “yes” or “no” answers were not feasible).

In order to help interviewees imagine concrete decision support tasks and situations in which a comprehensive SE-DSS might (or might not) be helpful, we offered three scenarios. A scenario consisted of a common part that served for setting the scene of management decision-making (i.e., what kind of information can be obtained, what is the basis for decision support, what is not available), and specific parts linked to the following: (1) quality manager, (2) project manager, and (3) product manager.

The questionnaire was developed in collaboration with an expert in cognitive psychology and was based on experience gained in previous projects (cf. for example [7]). The questions not only aimed at eliciting requirements from potential future users of a comprehensive SE-DSS, but also to substantiate the validity of the scenarios offered to the interviewees.

The interviews were conducted as follows. Interviewees received the common part of the scenario description and two role-specific scenario descriptions a couple of days prior to the interview. When the interview started, first the role-specific scenario was presented to the interviewee. Then, the interviewee was asked to answer the questions from the perspective of the first role. When all questions related to the first role had been answered, the second role-specific scenario was presented to the interviewee, and the interviewee was asked to assume the second role and think about differences in the requirements for that role. $\frac{3}{4}$ of the time were assigned to the first role, $\frac{1}{4}$ to the second role. Eight of the nine interviews were recorded with an MP3 stick. In one case, a scribe recorded the interview on paper. All interviews lasted between 25 and 35 minutes.

Table 1. Scenario assignments to interviewees

Interviewee	A	B	C	D	E	F	G	H	J
Role	1	1	2	1	1	2	1	1	2
Role	2	3	3	2	3	3	2	3	3

Each interviewee was randomly assigned to two of the three specific roles (c.f. Table1; 1 = quality manager; 2 = project manager; 3 = product manager). The purpose of having different scenarios for three different management roles was to find out whether these differ in their user requirements. The set of questions was not sent to the interviewees in advance. Also, there was no

communication between interviewees about the content of the interviews while the study was conducted.

The procedure we used to aggregate and synthesize the answers given by the interviewees was inspired by the grounded theory approach [8]. We started the transcription with the first interview and the first question. Then we took the next interview and tried to find communalities and differences related to the first answer of the first question. If a similar answer was found, the counter of the first answer to the first question was set from 1 to 2. If no sufficient similarity was encountered, then the new answer from the second interview was added to the list of answers related to the first question. When all interviews were checked for question one, we repeated this procedure for question two, starting with the first interview. If an answer was found to be more related to another question, it was re-assigned to that question, following the procedure described above. After having processed all answers related to all questions, we double-checked that the aggregated and synthesized answers still represented sufficiently well the set of answers originally provided by the interviewees.

In addition to counting the occurrence of similar answers, a binary ranking was made: the interviewee (H) explicitly or intuitively expressed high importance of the response to the question, (M) either explicitly ranked it as medium important or did not clearly rank it as highly important. The process of aggregation and ranking resulted in Tables 2 to 6.

Table 2. Motivation for DSS usage (question 1)

Why would you use a DSS in the given situation?			
		H	M
1.1	To get faster, broader, independent and empirically validated information about effectiveness and efficiency of a particular SE technology.	2	3
1.2	To answer the question: Which SE technique is most efficient / effective in a particular context (organization, process, product, documents)?	1	2
1.3	To get an overview on the existing techniques.	1	1
1.4	To get quantitative information (costs, quality level, defect reduction rate) about effectiveness and efficiency of a SE technique; people often tend to deliver qualitative information.	3	
1.5	Access to external information, which are otherwise not easy to get	2	2

Table 3. Benefits for organizational improvement management (question 2)

How could a DSS contribute to organizational improvement management?			
		H	M
2.1	By connecting it with the internal software improvement management (One interviewee gave the hint that this connection will only work in one direction, i.e. data will be imported from the DSS into the organizational improvement management but not vice versa)	3	
2.2	By enhancing experience management	1	1
2.4	By benefiting from experience of others	3	

Table 4. Interaction preferences (question 3)

Two alternative interaction strategies. Table 4. Similar to a search engine but more specialized.			
2. Iterative refinement of the solution area by user model based interaction. Which strategy would you prefer, and why?		H	M
3.1	A combination of the alternatives is preferable	3	4
3.2	Transparency is important: Why did I get this result set? Access to the full set should be possible	4	
3.3	Not answering lots of questions, but fill in a template with check-boxes	2	
3.4	Especially in case of a huge result set, the second alternative becomes more attractive	3	1
3.5	Guidance for reducing the result set (e.g., use the context to reduce result set)	3	
3.6	Interaction has to be goal/problem oriented	1	

Table 5. Types of information needed (questions 4+5)

Results from empirical studies can be described and aggregated differently. Which information should be provided by the DSS?			
		H	M
4.1	Which techniques are available (information on a highly aggregated level)?	3	1
4.2	How effective/efficient is a certain technique with respect to which quality aspect?	3	2
4.3	Description of the process in which a SE technique shall be applied	1	
4.4	Costs for introducing/applying the SE technique	2	2
4.5	To get information about the impact a single SE technique has on the whole development process	1	
4.6	Information that allows for conclusions about the validity of empirical results associated with a particular SE technique	3	2
4.7	Context information (kind of system, programming language, process step)	2	1
4.8	Preconditions that have to be fulfilled prior to the application of the SE technique (e.g., skills, kind of documents available)	1	1

The fifth question was used to prioritize different types of information obtainable from controlled experiments. Answers to this question were aggregated with responses given to the fourth question (cf. Table 5).

Table 6. Presentation preferences (question 6)

How should the information be presented?			
		H	M
6.1	Profile for each SE technique (details on request)	1	1
6.2	Aggregated information in multiple graphical presentation	5	1
6.3	Easy-to-understand, self-explaining diagram	6	1
6.4	Easy-to-understand, self-explaining table	7	1
6.5	Executive management summary		1

The seventh question was not intended to elicit new requirements but to confirm the relevance of our scenarios, and to identify new/other application areas for a comprehensive SE-DSS. Since the answers were not used for requirement elicitation, we omit the related table here. The relevance of the scenarios was confirmed. In addition, the answers confirm findings from question two, but on a more general level. For example, it was mentioned that a comprehensive SE-DSS could be used to

educate new employees, or store (and maintain) project experience. Additionally, the available information might be used to focus future studies on SE technology effectiveness/efficiency, and thus help improve the coordination of empirical research.

4. Additional Requirements

Additional requirements that relate to the needs of content (data) contributors, administrators, and the sponsor of the comprehensive SE-DSS were derived from lessons learned we gained with setting-up and running web-based repositories [3, 4, 7]. Table 7 lists these additional requirements. Requirements that emerge from the envisioned comprehensive DS method will also impact the internal system functionality, but are not considered here.

Table 7. Additional Requirements

#	
AR1	Support for distributed contribution
AR2	Support for distributed quality management
AR3	Multi-role management
AR4	Multiple cross-linking of content items

(AR1) It must be possible for the research community to contribute with new studies on SE technologies.

(AR2) AR1 requires at least some degree of quality assurance (QA), e.g., by establishing a QA board that is in charge of approving new contributions.

(AR3) AR1 and AR2 lead to at least two more different roles, i.e., contributor and QA.

(AR4) To enable the drawing of a landscape that visualizes the relationship between empirical studies on the effectiveness and efficiency of SE technologies.

5. Structured List of Requirements

Based on the two sources of requirements, described above, we ordered the requirements according to the standard three-tier architecture, and according to Ruhe's generic requirements categories.

Table 8 Requirements

#	User Interface	Reference
UI1	Support for several kinds of graphical / textual presentations	6.1, 6.2, 6.3, 4
UI2	Low interaction, easy access	3.1, 3.4, 1.4
UI3	Goal-oriented interaction support	3.6, 3.5
UI4	Alternative interaction modes	3.1, 3.4, 3.3
Presentation		
PR1	Transparency of decision process (reduction of alternatives, priorities)	2.1
PR2	Goal/problem-oriented aggregation of information	6.1, 6.2, 4.1, 4.3, 4.5, 4.6, 6.5
PR3	Understandable, self-explaining	6.3, 6.4
PR4	Presentation in diagrams, tables, text	6.1-6.5
Content		

CO1	Effectiveness/efficiency with respect to quality aspect	4.2
CO2	Costs for introduction/applying the technique	1.4, 4.4
CO3	Preconditions that have to be fulfilled prior to the application of the technique	4.8
CO4	Context information	1.2
CO5	Structured meta information for the content	2.3, 4
Experience Management		
EM1	Support for distributed contribution	AR1
EM2	Support for distributed quality assurance (distributed content management)	AR2
EM3	Support for export of repository data to organizational improvement management systems	2.1, 2.2
EM4	Multi-role management	AR3
Repository		
RE1	Cross-linking of experience items	AR4
RE2	Case-oriented storing	4.5, 4.6, 2.4

Table 9. Mapping to Ruhe's idealized requirements

Ruhe's Framework [1]	Specific user requirements for comprehensive SE-DSS
(R1) Knowledge, model and experience management	EM1-EM3, CO1-CO5
(R2) Integration into organization	EM3-EM4, RE1
(R3) Process orientation	CO3-CO5, PR1-PR2, EM4
(R4) Process modeling and simulation	CO3-CO5
(R5) Negotiation	--
(R6) Presentation and explanation	PR1-PR5
(R7) Analysis and decision	PR1-PR2, RE1-RE2
(R8) Intelligence	RE1-RE2

Table 8 lists the requirements derived from the pilot study and combines them with the additional requirements. The set of requirements is grouped into five categories: user interface (UI), presentation (PR), content (CO), experience management (EM), and repository (RE). The first two categories correspond to the first layer of the standard three-tier architecture, the third category corresponds to the second layer, and the fourth and fifth categories correspond to the third layer. Column three of provides for each requirement the reference to related aggregated answers or additional requirements.

Table 9 shows the mapping of the requirements for the comprehensive SE-DSS to the framework "idealized" requirements (R1-R9) suggested by Ruhe [1]. The instantiation depends on our concrete problem topic, i.e., comprehensive SE technology selection, and usage scenarios, i.e., on-line, individual and strategic decision support for project, quality, and product management. One lesson we learned was that the framework was sufficiently generic to incorporate all of our specific requirements.

6. Summary and Future Work

In this paper we have presented the requirements of a web-based tool for comprehensive decision-making in support of SE technology selection. The requirements

were collected from a literature survey and from structured interviews with research managers.

Besides the identification of requirements, the research yielded the following results: All of the interviewees accepted the pre-defined scenarios as being relevant and practical, none had difficulties with understanding. We interpret this finding to support the construct validity of our measurement instrument (scenario-based structured interviews).

Surprisingly, we did not find much difference between management roles. Apart from prioritization of content presentation (question 5), the answers given were very similar, no matter which specific role was assigned to an interviewee. At the moment, it is not fully clear whether this indicates that differences between roles are not as large as we originally expected, or whether the answers given by the interviewees were too strongly influenced by the way role-specific scenarios were presented to them. Also, the subjects might not be fully representative for the specified roles due to the nature of their work in research environments, which is probably not as strongly focused on actual (and mostly short-term) decision-making within software projects.

Future work is dedicated to the incremental development of the comprehensive SE-DSS. At each stage, the underlying method and the resulting tool will be evaluated through controlled experiments and surveys among experts from academia and industry. Issues to be evaluated include effectiveness and efficiency of the method and tool support, as well as validity of the delivered information and completeness of the database.

References

- [1] Ruhe, G.: "Software Engineering Decision Support – A new paradigm for Learning Software Organizations". In: *Proc. Workshop. Learning Software Organizations*, Springer, 2003.
- [2] Ruhe, G.: "Software Engineering Decision Support: Methodology and Applications". In: *Innovations in Decision Support Systems* (Ed. by Tonfoni and Jain), International Series on Advanced Intelligence Volume 3, 2003, pp 143-174.
- [3] Jedlitschka, A.; Pfahl, D. and Bomarius, F.: "A Framework for Comprehensive Experience-based Decision Support for Software Engineering Technology Selection"; In *Proc. of Intern. Conf. SEKE 2004*. Banff, Canada, 2004
- [4] Jedlitschka, A.; Ciolkowski, M.: "Towards Evidence in Software Engineering"; In *Proc. of ACM/IEEE ISESE 2004*, Redondo Beach, California, August 2004, IEEE CS, 2004.
- [5] Conradi, R.; Wang, A.I (Eds.): *Empirical Methods and Studies in Software Engineering – Experiences from ESERNET*; Springer LNCS 2765, 2003.
- [6] Biffi, S.; Halling, M.: "A Knowledge Management Framework to Support Software Inspection Planning", in [9]
- [7] Jedlitschka, A.; Nick, M.: "Software Engineering Knowledge Repositories"; in [5] pp.55-80
- [8] Strauss, A. & Corbin, J.: *Basics of Qualitative Research. Techniques and Procedures for Developing Grounded Theory*. 2nd ed. Thousand Oaks: Sage, 1998.
- [9] Aurum, A.; Jeffery, R.; Wohlin, C.; Handzic, M. (Eds): *Managing Software Engineering Knowledge*; Springer-Verlag; Berlin 2003

Reviewers'

A

Silvia Teresita Acuna
Anneliese K. Amschler Andrews
Juan Carlos Augusto
Aybuke Aurum

B

Piefrancesco Bellini
Sami Beydeda
Alessandro Bianchi
Frank Bomarius

C

Kai-Yuan Cai,
Zhining Cao,
Rosa M. Carro,
Alejandra Cechich,
María Dolores Vargas Cerdán,
Yurong Chen
William Chu
Paolo Ciancarini
Oscar Corcho
Patricia Costa

D

Feras T. Dabous
Angélica de Antonio
John Debenham
Yi Deng
Vincenzo Deufemia
Oscar Dieste
Paolo Donzelli
Toncan Duong
Schahram Dustdar

E

Hakan Erdogmus

F

Pascal Fenkam
Xavier Ferre
Filomena Ferrucci
Andres Flores
Rita Francese
Alfonso Fuggetta

G

Shu Gao
Carlo Ghezzi
Marisol Giardina
Athula Ginige
Haitao Gong

Carmine Gravino
Volker Gruhn
John Grundy
Thomas Gschwind

H

Mariele Hagen
Aaron Hector
Bayu Hendradjaya
Pilar Herrero
Lorin Hochstein
Siv Hilde Houmb
Mao Lin Huang

I

Hiroshi Igaki
José Antonio Macías Iglesias Hajimu Iida

J

Letizia Jaccheri
Zhi Jin
Kanta Jiwnani
Natalia Juristo

K

André Köhler
Jun Kong
Serguei Krivov
Cat Kutay

L

Guojun Li
Jingzhou Li
Sheldon X. Liang
Hong-Xin Lin
Huimin Lin
Pdero Linares
Mikael Lindvall
Jiming Liu
Fabiola Lopez y Lopez
Andrea De Lucia
Luqi

M

Sergio Di Martino
Frank Maurer
Nelson Medinlla
Gonzalo Méndez
Abdallah Mohamed
Sandro Morasca
Ana M. Moreno
Jurgen Munch
Ming Muo

N

Lakshmi Narasimhan
Abhaya Nayak
Josef Nedstam
Paolo Nesi
An Ngo-The

O

Andrew O'Fallon
Mehmet Orgun
Alvaro Ortigosa
Thomas Østerlie

P

Luca Paolino
Orest Pilskalns
Martin Pinzger
Giuseppe Polese

Q

Yu Qian

R

Fethi Rabhi
Jaime Ramírez
Michael Richter
Michele Risi
Guenther Ruhe
Ioana Rus

S

Omolade Saliu
Marisa Sanchez
Maria-Isabel Sanchez-Segura
Maribel Sanchez-Segura
Walt Scacchi
Giuseppe Scanniello
Klaus Schmid
Indra Seher
Michele A. Shaw
John Shepherd
Phillip Sheu
Xiaochun Shi
Alejandro Sierra

Almudena Sierra-Alonso

Janice Singer
Guanglei Song
Lorna Stewart
Eleni Stroulia
Weixiang Sun
Magne Syrstad

T

Scott Tilley
Cora B. Excelente Toledo
Genny Tortora
Jeffrey Tsai

V

Sira Vegas
Maximiliano Paredes Velasco
Giuseppe Visaggio
Giuliana Vitiello

W

Qing Wang
Yingxu Wang
Christiane Gresse von Wangenheim
Richard Webber
Stefan Wermter
Xindong Wu

Y

Ying Yang
JingTao Yao
Yiyu Yao
Huilin Ye
InSeon Yoo
Huiqun Yu
Hairong Yu

Z

Guangcun Zhang
Kang Zhang
Xu Zhang
Haiyan Zhao
Liming Zhu
Xingquan Zhu

Authors' Index

A

Marzia Adorni, 74
 J. Ahn, 37
 Lynda Ait-Mahiedine, 508
 Boanerges Aleman-Meza, 490
 Reda Alhadj, 498
 C. F. Allgood, 167
 Julio César Alvarez, 435
 L. An, 431
 Anneliese Andrews, 129
 Giuliano Antoniol, 449
 João Araújo, 411
 Moysés de Araújo, 478
 Francesca Arcelli, 74
 Jocelyn Armarego, 421
 I. Budak Arpinar, 490
 Mikhail Auguston, 185
 Paolo Avesani, 306

B

Fabian B'uttner, 135
 K. S. Barber, 37
 Ken Barker, 498
 Márcio Barros, 19
 Márcio de O. Barros, 282
 Steve Battle, 98
 M. Baumgarten, 388
 Cinzia Bazzanella, 306
 Carlos Bento, 258
 Erik Berglund, 246
 Sami Beydeda, 104
 Konstantin Beznosov, 360
 Antonio Boccalatte, 45
 Frank Bomarius, 342
 Kaddour Boukerche, 457
 Imen Bourguiba, 252
 Barrett R. Bryant, 185
 Rebecca Buchheit, 80
 A.G. Büchner, 388
 Jim Buckley, 486
 Carol C. Burt, 185
 Y. Bychkov, 270

C

Ernest Cachia, 318
 Tony Cahill, 486
 Yuhong Cai, 276
 Osvaldo Cairó, 435
 Coral Calero, 392
 Gerardo Canfora, 57

Fei Cao, 185

Daniel Amaral Cardoso, 396
 Paulo Carreiro, 258
 Luigi Cerulo, 57
 Christine W. Chan, 86
 Henry Chang, 431
 Bo Chen, 300
 Chien-Hsien Chen, 384
 Wei Cheng, 402
 J. Chung, 431
 C. M. Chewar, 167
 Rem Collier, 25
 Damien Conroy, 486
 Devin Cook, 179
 Kendra Cooper, 360
 Alexandre Correa, 294
 Hélio R. Costa, 282
 G. Costagliola, 439
 Adrien Coyette, 192
 José A. Cruz-Lemus, 238
 Davor Cubrani, 92

D

Xiaoling Dai, 276
 Alexandre Dantas, 19
 R. Delmonico, 372
 Yi Deng, 360, 416
 Marcos Vinícius Pinheiro Dib, 396
 Tung Do, 192

E

Armin Eberlein, 324
 Raimund K. Ege, 155
 Karin Ericsson, 205

F

Ricardo de Almeida Falbo, 474
 Christos Faloutsos, 80
 Behrouz H. Far, 324
 Fausto Fasano, 31, 453
 Stéphane Faulkner, 192
 A. Felfernig, 148
 Han Fengyan, 13
 Jos L. Ferreira, 258
 F. Ferrucci, 439
 Stephan Flake, 161
 T.W. Fox, 142
 B.J. Fox, 142
 Rita Francese, 31, 453
 Bernd Freimut, 264

G

Michel Gagnon, 449
Kehan Gao, 220
Shu Gao, 360
Marcela Genero, 238
Daniel M. German, 336
Jeewani Anupama Ginige, 445
Athula Ginige, 445, 466
Martin Gogolla, 135
Paulo Gomes, 258
Andrea Gozzi, 45
T. Graser, 37
C. Gravino, 439
D. Greer, 503
Alberto Grosso, 45
Volker Gruhn, 104
John Grundy, 276
N. Gujral, 37

H

Chris Halaschek, 490
Babak Hamidzadeh, 494
Xudong He, 360, 416
Dennis Heimbigner, 470
Guillermo Nudelman Hess, 366
Abram Hindle, 336
Reginald L. Hobbs, 330
John Hosking, 276
Zhaoxia Hu, 213
J.G. Hughes, 388

I

Benedetto Intrigila, 7
Cirano Iochpe, 366
Lee Iverson, 494

J

J. H. Jahnke, 270
Dietmar Jannach, 110
Stan Jarzabek, 68
Olga Jaufman, 264
Andreas Jedlitschka, 342, 513
Jun-Jang Jeng, 431
Norman Jordan, 336
Vedang H. Joshi, 226
Natalia Juristo, 378

K

Mira Kajko-Mattsson, 205
Aditya Kalyanpur, 98
Tahar Khammaci, 346
Ridha Khedri, 252
Taghi M. Khoshgoftaar, 220, 226

T. Klinger, 372

Andrew Knight, 129
Yoshitake Kobayashi, 350
Manuel Kolp, 192
Philip Koopman, 80
Gerold Kreutler, 110

L

D. N. Lam, 37
Chiou Peng Lam, 421
Anders Larsson, 246
Anna Rita Laurenzi, 7
Seok Won Lee, 117
Dominic Letarte, 449
Honglian (Elena) Li, 62
Weigang Li, 396
Yun Li, 402
Arne Lindow, 135
Dong Liu, 324
Wei Liu, 402
Zongtian Liu, 402
Hakim Lounis, 457, 508
Andrea De Lucia, 31, 453
Michael Lutz, 482

M

Mamoru Maekawa, 350
Alba Cristina Magalhães, 396
Carmen Maidantchik, 427
Ioakim (Makis) Marmaridis, 445
Valerie Maxville, 421
D. Scott McCrickard, 167
Alves de Melo, 396
Ettore Merlo, 449
Mark Micallef, 318
Vanessa Mirzaee, 494
Lian Mo, 416
Mariano Montoni, 427
Ana Moreira, 411
Ana M. Moreno, 378
Malek Mouhoub, 406
Ethan V. Munson, 288
Gail C. Murphy, 92

N

Ken Nakayama, 350
Tai Nguyen, 192
Tien N. Nguyen, 288

O

Rory V O'Connor, 312
Andrew O'Fallon, 129
Gregory O'Hare, 25
José A. Olivas, 238
Rocco Oliveto, 453

Andrew M. Olson, 185
Sergio Orefice, 7
Mehmet Orgun, 51
Mourad Oussalah, 346

P

Julian Padget, 98
Paulo Paiva, 258
Daniel Jiménez Pastor, 98
W.D. Patterson, 388
Giuseppe Della Penna, 7
Francisco C. Pereira, 258
Anna Perini, 306
Dietmar Pfahl, 342, 513
Shari Lawrence Pfleeger, 1
Mario Piattini, 238, 392
Orest Pilskalns, 129

R

Oliver Radfelder, 135
Claudia Raibulet, 74
Rajeev R. Raje, 185
B. Ray, 372
Orna Raz, 80
Jörg Rech, 462
David C. Rine, 117
Dieter Rombach, 2
Francisco P. Romero, 238
Colm Rooney, 25
Shen Ru, 68
Ioana Rus, 264
Sharon Ryan, 312

S

Samira Sadaoui, 300, 406
Houari A. Sahraoui, 457
Isabel Sánchez, 378
Gowtham Sannapareddy, 490
P. Santhanam, 372
Gleison Santos, 427
J. Schiefer, 431
Nuno Seco, 258
Indra Seher, 466
Manuel Serrano, 392
Theodorus Eric Setiadi, 350
Sol M. Shatz, 213
Mary Shaw, 80
Amit Sheth, 490
George Shi, 498
Adel Smeda, 346
Spencer Smith, 384
Peter Stanski, 173

Nenad Stojanovic, 232
Eleni Stroulia, 123
Kalaivani Subramaniam, 324
Amrudee Sukpan, 406
Angelo Susi, 306
Zsafia Szalkai, 205

T

Cheng Thao, 288
Francesco Tisato, 74
G. Tortora, 439
Genoveffa Tortora, 31
Guilherme H. Travassos, 282
Luigi Troiano, 57

V

Christian Vecchiola, 45
G. Vitiello, 439

W

William W. Wadge, 62
Shahtab Wahid, 167
Cláudia Werner, 19, 294
Marselina Wiharto, 173
Qiang Wu, 402

X

Yudong Xiao, 220
Wang Xin, 13
Zhenchang Xing, 123
Liyin Xue, 51

Y

Li Yang, 155
J.T. Yao, 199
Y.Y. Yao, 199
Huiqun Yu, 155, 360, 416

Z

Lotfi A. Zadeh, 3
L. Zheng, 431
Hongyu Zhang, 68
Kang Zhang, 51
Du Zhang, 179
Gang Zhao, 354
Wei Zhao, 185
Y. Zhao, 199
Qin Zheng, 13
Sun Zhenxin, 68



Call for Papers

2005 International Conference of Software Engineering and Knowledge Engineering

July 14 to 16, 2005

Taipei, Taiwan, Republic of China



Taipei 101

SCOPE

The conference aims at bringing together experts in software engineering and knowledge engineering to discuss on relevant results in either software engineering or knowledge engineering or both. Special emphasis will be put on the transference of methods between both domains.

TOPICS

Solicited topics include, but are not limited to:

Adaptive Systems
Artificial Intelligence Approaches to Software Engineering
Automated Reasoning
Automated Software Design and Synthesis
Automated Software Specification
Component-Based Software Engineering
Computer-Supported Cooperative Work
Databases
Design Methods
Education and Training
Electronic Commerce
Formal Methods
Human-Computer Interaction
Industrial Applications
Integrity, Security, and Fault Tolerance
Knowledge Acquisition
Knowledge-Based and Expert Systems
Knowledge Representation and Retrieval
Knowledge Engineering Tools and Techniques
Knowledge Visualization
Learning Software Organization
Measurement and Empirical Software Engineering
Meta-CASE
Mobile Data Accesses
Multimedia and Hypermedia Software Engineering
Object-Oriented Technology
Ontologies and Methodologies
Patterns and Frameworks
Process and Workflow Management
Programming Languages and Software Engineering
Program Understanding
Reflection and Metadata Approaches
Reliability
Requirements Engineering
Reverse Engineering
Soft Computing
Software Architecture
Software Domain Modeling and Meta-Modeling
Software Engineering Decision Support
Software Maintenance and Evolution
Software Process Modeling
Software Quality

Software Reuse
System Applications and Experience
Time and Knowledge Management
Tools
Tutoring, Help, Documentation Systems
Uncertainty Knowledge Management
Validation and Verification
Web-Based Knowledge Management
Web-Based Tools, Systems, and Environments
Web and Data Mining

STEERING COMMITTEE

Vic Basili, University of Maryland
Bruce Buchanan, University of Pittsburgh
Shi-Kuo Chang, University of Pittsburgh, Pittsburgh
C. V. Ramamoorthy, University of California, Berkeley

GENERAL CO-CHAIRS

J. S. Ke, Institute for Information Industry, Taiwan
B. S. Lin, Industry Technologies Research Institute, Taiwan
A. C. Liu, Feng-Chia University, Taiwan

INFORMATION FOR AUTHORS

Paper deadline is **March 1, 2005**. Papers must be written in English. An electronic version (Postscript, PDF, or MS Word format) of the full paper should be submitted using the following URL: <http://conf.ksi.edu/seke05/submit/SubmitPaper.php>. Please use Internet Explorer as the browser. Manuscript must include a 200-word abstract and no more than 6 pages of IEEE double column text (include figures and references).

INFORMATION FOR REVIEWERS

Papers submitted to SEKE'05 will be reviewed electronically. The users (webmaster, program chair, reviewers...) can login using the following URL: <http://conf.ksi.edu/seke05/review/pass.php>. If you have any questions or run into problems, please send e-mail to: seke@ksi.edu.

SEKE'2005 Conference Secretariat
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076 USA
Tel: 847-679-3135
Fax: 847-679-3166
E-mail: seke@ksi.edu

**Proceedings of the
Sixteenth International
Conference on Software
Engineering & Knowledge
Engineering**

**Sponsored by
Knowledge Systems Institute
iCORE
University of Calgary**

**Printed by
Knowledge Systems Institute
3420 Main Street
Skokie, Illinois 60076
(847) 679-3135
info@ksi.edu
www.ksi.edu**



2004 SEKE
**Alberta, Canada
June 20 to June 24, 2004**