ICSOFT 2006

Proceedings of the
First International Conference on
Software and Data Technologies

Volume 1

Setúbal, Portugal

September 11 – 14, 2006

Organized by
INSTICC – Institute for Systems and Technologies of Information,
Control and Communication

Sponsored by
Enterprise Ireland
Polytechnic Institute of Setúbal

In Cooperation with
Object Management Group (OMG)

Hosted by
School of Business of the Polytechnic Institute of Setubal
Copyright © INSTICC – Institute for Systems and Technologies of Information, Control and Communication
All rights reserved

Edited by Joaquim Filipe, Boris Shishkov and Markus Helfert

Printed in Portugal
Depósito Legal: 246927/06

http://www.icsoft.org
secretariat@icsoft.org
BRIEF CONTENTS

BRIEF CONTENTS ................................................................. III
KEYNOTE LECTURES ............................................................. IV
TUTORIAL ........................................................................ IV
ORGANIZING AND STEERING COMMITTEES .................... V
PROGRAM COMMITTEE ....................................................... VII
AUXILIARY REVIEWERS ..................................................... IX
SELECTED PAPERS BOOK ............................................... X
FOREWORD ..................................................................... XI
CONTENTS ....................................................................... XIII
KEYNOTE LECTURES

Leszek A. Maciaszek
Macquarie University
Australia

Juan Carlos Augusto
University of Ulster at Jordanstown
U.K.

Tom Gilb
Norway

Dimitris Karagiannis
University of Vienna
Austria

Brian Henderson-Sellers
University of Technology
Australia

Marten J. van Sinderen
University of Twente
The Netherlands

TUTORIAL

Tom Gilb
Norway
ORGANIZING AND STEERING COMMITTEES

Conference Chair
Joaquim Filipe, INSTICC/Polytechnic Institute of Setúbal, Portugal

Program Co-chairs
Markus Helfert, Dublin City University, Ireland
Boris Shishkov, University of Twente, The Netherlands

Proceedings Production
Paulo Brito, INSTICC, Portugal
Marina Carvalho, INSTICC, Portugal
Helder Coelhas, INSTICC, Portugal
Bruno Encarnação, INSTICC, Portugal
Vitor Pedrosa, INSTICC, Portugal

Webdesign and Graphics Production
Marina Carvalho, INSTICC, Portugal

Secretariat and Webmaster
Mónica Saramago, INSTICC, Portugal
PROGRAM COMMITTEE

Hamideh Afsarmanesh, University of Amsterdam, The Netherlands
Jacky Akoka, CNAM & INT, France
Tsanka Angelova, Unicord Ltd, Bulgaria
Keijiro Araki, Kyushu University, Japan
Lora Aroyo, Eindhoven University of Technology, The Netherlands
Colin Atkinson, University of Mannheim, Germany
Juan Carlos Augusto, University of Ulster at Jordanstown, U.K.
Elisa Baniassad, Chinese University of Hong Kong, China
Mortaza S. Bargh, Telematica Instituut, The Netherlands
Joseph Barjis, Georgia Southern University, U.S.A.
Noureddine Belkhatir, LSR-IMAG University of Grenoble, France
Fevzi Belli, University Paderborn, Germany
Alexandre Bergel, Trinity College, Ireland
Mohamed Bettaz, Philadelphia University, Jordan
Robert Biddle, Carleton University, Canada
Maarten Boasson, University of Amsterdam, The Netherlands
Wladimir Bodrow, University of Applied Sciences Berlin, Germany
Marcello Bonsangue, LIACS - Leiden University, The Netherlands
Jonathan Bowen, London South Bank University, U.K.
Mark van den Brand, Technical University of Eindhoven, The Netherlands
Lisa Brownsword, Software Engineering Institute, U.S.A.
Barrett Bryant, University of Alabama at Birmingham, U.S.A.
Cinzia Cappiello, Politecnico di Milano, Italy
Antonio Cerone, UNU-IIST, China
W. K. Chan, Hong Kong University of Science and Technology, China
Kung Chen, National Chengchi University, Taiwan
Samuel Chong, Accenture, U.K.
Chih-Ping Chu, National Cheng Kung University, Taiwan
Peter Clarke, Florida International University, U.S.A.
Rolland Colette, University Paris I Pantheon Sorbonne, France
Alfredo Cuzzocrea, University of Calabria, Italy
Bogdan Czejdo, Loyola University, U.S.A.
David Deharbe, UFRN/DIMAp, Brazil
Serge Demeyer, University of Antwerp, Belgium
Steve Demurjian, University of Connecticut, U.S.A.
Nikolay Diakov, CWI, The Netherlands
Jan L. G. Dietz, Delft University of Technology, The Netherlands
Jin Song Dong, National University of Singapore, Singapore
Brian Donnellan, National University of Ireland, Ireland
Jürgen Ebert, University Koblenz, Germany
Paul Ezhilchelvan, University of Newcastle, U.K.
Behrouz Far, University of Calgary, Canada
Bernd Fischer, University of Southampton, U.K.
Gerald Gannod, Arizona State University, U.S.A.
Jose M. Garrido, Kennesaw State University, U.S.A.
Dragan Gasevic, Simon Fraser University, Canada
Nikolaos Georgantas, INRIA Rocquencourt, France
Paola Giannini, Università del Piemonte Orientale, Italy
Paul Gibson, National University of Ireland, Maynooth, Ireland
Wolfgang Grieskamp, Microsoft Research, U.S.A.
Daniela Grigori, University of Versailles, France
Klaus Grimm, Daimlerchrysler Ag, Germany
Rajiv Gupta, University of Arizona, U.S.A.
Tibor Gyimothy, University of Szeged, Hungary
Naohiro Hayashibara, Tokyo Denki University False, Japan
Jang Eui Hong, Chungbuk National University, Korea
Shinichi Honiden, National Institute of Informatics, Japan
PROGRAM COMMITTEE (CONT.)

Ilian Ilkov, IBM Nederland B.V, The Netherlands
Ivan Ivanov, SUNY Empire State College, U.S.A.
Tuba Yavuz Kahveci, University of Florida, U.S.A.
Krishna Kavi, University of North Texas, U.S.A.
Khaled Khan, University of Western Sydney, Australia
Roger King, University of Colorado, U.S.A.
Christoph Kirsch, University of Salzburg, Austria
Paul Klint, Centrum voor Wiskunde en Informatica (CWI) en University of Amsterdam, The Netherlands
Alexander Knapp, Ludwig-Maximilians-Universität München, Germany
Mieczyslaw Kokar, Northeastern University, U.S.A.
Michael Kölling, University of Kent, U.K.
Dimitri Konstantas, University of Geneva, Switzerland
Jens Krinke, FernUniversität in Hagen, Germany
Tei-Wei Kuo, National Taiwan University, Taiwan
Rainer Koschke, University of Bremen, Germany
Eitel Lauria, Marist College, U.S.A.
Insup Lee, University of Pennsylvania, U.S.A.
Kuan-Ching Li, Providence University, Taiwan
Panos Linos, Butler University, U.S.A.
Shaoying Liu, Hosei University, Japan
Zhiming Liu, UNU-IIST, China
Andrea De Lucia, Università di Salerno, Italy
Christof Lutteroth, University of Auckland, New Zealand
Broy Manfred, Institut für Informatik, TU München, Germany
Tiziana Margaria, University of Göttingen, Germany
Johannes Mayer, Ulm University, Germany
Fergal McCaffery, University of Limerick, Ireland
Hamid Mcheick, University of Quebec at Chicoutimi, Canada
Prasenjit Mitra, The Pennsylvania State University, U.S.A.
Dimitris Mitrakos, Aristotle University of Thessaloniki, Greece
Roland Mittermeier, Universitäet Klagenfurt, Austria
Birger Moller-Pedersen, University of Oslo, Norway
Mattia Monga, Università degli Studi di Milano, Italy
Aldo De Moor, Vrije Universiteit Brussel, Belgium
Peter Müller, ETH Zurich, Switzerland
Paolo Nesi, University of Florence, Italy
Elisabetta Di Nitto, Politecnico di Milano, Italy
Alan O’Callaghan, De Montfort University, U.K.
Rory O’Connor, Dublin City University, Ireland
Claus Pahl, Dublin City University, Ireland
Witold Pedrycz, University of Alberta, Canada
Massimiliano Di Penta, RCOST - University of Sannio, Italy
Steef Peters, Vrije Universiteit Amsterdam, The Netherlands
Mario Piattini, University of Castilla-La Mancha, Spain
Arnd Poetzsch-Heffter, University of Kaiserslautern, Germany
Andreas Polze, Hasso-Plattner-Institute, Univ. Potsdam, Germany
Christoph von Praun, IBM Research, U.S.A.
Jolita Ralyte, University of Geneva, Switzerland
Juan Fernandez Ramil, The Open University, U.K.
Anders P. Ravn, Aalborg University, Denmark
Marek Reformat, University of Alberta, Canada
Arend Rensink, University of Twente, The Netherlands
Stefano Russo, Federico II University of Naples, Italy
Shazia Sadiq, The University of Queensland, Australia
Kristian Sandahl, Linköping University, Sweden
Bradley Schmerl, Carnegie Mellon University, U.S.A.
Andy Schürr, Darmstadt University of Technology, Germany
Isabel Seruca, Universidade Portucalense, Portugal
COMMITTEE (CONT.)

Marten van Sinderen, University of Twente, The Netherlands
Joao Sousa, Carnegie Mellon University, U.S.A.
George Spanoudakis, City University, U.K.
Peter Stanchev, Kettering University, U.S.A.
Larry Stapleton, ISOL Research Centre, Ireland
Stoicho Stoichev, Technical University-Sofia, Bulgaria
Kevin Sullivan, University of Virginia, U.S.A.
Junichi Suzuki, University of Massachusetts, U.S.A.
Ramayah Thurasamy, Universiti Sains Malaysia, Malaysia
Yasar Tonta, Hacettepe University, Turkey
Yves Le Traon, France Télécom R&D, France
Enrico Vicario, University of Florence, Italy
Bing Wang, University of Hull, U.K.
Kun-Lung Wu, IBM Watson Research, U.S.A.
Hongwei Xi, Boston University, U.S.A.
Haiping Xu, University of Massachusetts Dartmouth, U.S.A.
Hongji Yang, De Montfort University, U.K.
Yunwen Ye, University of Colorado, U.S.A.
Yun Yang, Swinburne University, Australia
Gianluigi Zavattaro, University of Bologna, Italy
Xiaokun Zhang, Athabasca University, Canada
Jianjun Zhao, Shanghai Jiao Tong University, China
Hong Zhu, Oxford Brookes University, U.K.
Andrea Zisman, City University, U.K.

AUXILIARY REVIEWERS

Alessandro Aldini, Università degli Studi di Urbino, Italy
Pete Andras, University of Newcastle, U.K.
Xiaoshan Li, UNU-IIST, China
Shih-Hsi Liu, University of Alabama at Birmingham, U.S.A.
Michele Pinna, University of Bologna, Italy
Riccardo Solmi, University of Bologna, Italy
Hongli Yang, UNU-IIST, China
Chengcui Zhang, University of Alabama at Birmingham, U.S.A.
Liang Zhao, UNU-IIST, China
Wei Zhao, University of Alabama at Birmingham, U.S.A.
A number of selected papers presented at ICSOFT 2006 will be published by Springer, in a book entitled Software and Data Technologies. This selection will be done by the conference chair and program co-chairs, among the papers actually presented at the conference, based on a rigorous review by the ICSOFT 2006 program committee members.
FOREWORD

This volume contains the proceedings of the first International Conference on Software and Data Technologies (ICSOFT 2006), organized by the Institute for Systems and Technologies of Information, Communication and Control (INSTICC) in cooperation with the Object Management Group (OMG), sponsored by Enterprise Ireland and the Polytechnic Institute of Setúbal and hosted by the School of Business of the Polytechnic Institute of Setubal.

The purpose of this conference is to bring together researchers, engineers and practitioners interested in information technology and software development. The conference tracks are “Software Engineering”, “Information Systems and Data Management”, “Programming Languages”, “Distributed and Parallel Systems” and “Knowledge Engineering”.

Software and data technologies are essential for developing any computer information system, encompassing a large number of research topics and applications: from programming issues to the more abstract theoretical aspects of software engineering; from databases and data-warehouses to management information systems and knowledge-base systems; Distributed systems, ubiquity, data quality and other related topics are included in the scope of ICSOFT.

ICSOFT 2006 received 187 paper submissions from more than 39 countries in all continents. To evaluate each submission, a double blind paper evaluation method was used: each paper was reviewed by at least two internationally known experts from ICSOFT Program Committee. Only 23 papers were selected to be published and presented as full papers, i.e. completed work (8 pages in proceedings / 30’ oral presentations), 44 additional papers, describing work-in-progress, were accepted as short paper for 20’ oral presentation, leading to a total of 67 oral paper presentations. There were also 26 papers selected for poster presentation. The full-paper acceptance ratio was thus 12%, and the total oral paper acceptance ratio was 35%.

In its program ICSOFT includes a panel to discuss the future of software development, by six distinguished world-class researchers; furthermore, the program is enriched by one tutorial and six keynote lectures. These high points in the conference program, involving top researchers worldwide, experts in different knowledge areas, have definitely contributed to reinforce the overall quality of the conference.

The program for this conference required the dedicated effort of many people. Firstly, we must thank the authors, whose research and development efforts are recorded here. Secondly, we thank the members of the program committee and the additional reviewers for their diligence and expert reviewing. I would like to personally thank the Program Chairs, namely Boris Shishkov and Markus Helfert, for their important collaboration. The local organizers and the secretariat have worked hard to provide smooth logistics and a friendly environment, so we must thank them all and especially Mónica Saramago for her patience and diligence in answering many emails and solving all the problems. Last but not least, we thank the invited speakers for their invaluable contribution and for taking the time to synthesize and prepare their talks.
A successful conference involves more than paper presentations; it is also a meeting place, where ideas about new research projects and other ventures are discussed and debated. Therefore, a social event including conference banquet was organized for the afternoon and evening of September 13 (Wednesday) in order to promote this kind of social networking.

We wish you all an exciting conference and an unforgettable stay in the lovely city of Setúbal. We hope to meet you again next year for the 2nd ICSOFT, in Barcelona (Spain), details of which will be shortly made available at http://www.icsoft.org.

Joaquim Filipe

INSTICC/Polytechnic Institute of Setúbal, Portugal
(Conference Chair)
CONTENTS

INVITED SPEAKERS

KEYNOTE LECTURES

ADAPTIVE INTEGRATION OF ENTERPRISE AND B2B APPLICATIONS
Leszhek A. Maciaszek

AMBIENT INTELLIGENCE: BASIC CONCEPTS AND APPLICATIONS
Juan Carlos Augusto

HOW TO QUANTIFY QUALITY: FINDING SCALES OF MEASURE
Tom Gilb

QUANTIFYING STAKEHOLDER VALUES
Tom Gilb

METAMODELS IN ACTION: AN OVERVIEW
Dimitris Karagiannis and Peter Höfferer

ENGINEERING OBJECT AND AGENT METHODOLOGIES
Brian Henderson-Sellers

ARCHITECTURAL STYLES IN SERVICE ORIENTED DESIGN
Marten J. van Sinderen
# Programming Languages

**FULL PAPERS**

**ON STATE CLASSES AND THEIR DYNAMIC SEMANTICS**  
Ferruccio Damiani, Elena Giachino, Paola Giannini and Emanuele Cazzola  
Page 5

**SOFTWARE IMPLEMENTATION OF THE IEEE 754R DECIMAL FLOATING-POINT ARITHMETIC**  
Marius Cornea, Cristina Anderson and Charles Tsen  
Page 13

**FROM STATIC TO DYNAMIC PROCESS TYPES**  
Franz Puntigam  
Page 21

**ASPECTBOXES – CONTROLLING THE VISIBILITY OF ASPECTS**  
Alexandre Bergel, Robert Hirschfeld, Siobhán Clarke and Pascal Costanza  
Page 29

**SHORT PAPERS**

**ON ABILITY OF ORTHOGONAL GENETIC ALGORITHMS FOR THE MIXED CHINESE POSTMAN PROBLEM**  
Hiroshi Masuyama, Tetsuo Ichimori and Toshihiko Sasama  
Page 39

**ZÁS - ASPECT-ORIENTED AUTHORIZATION SERVICES**  
Paulo Zenida, Manuel Menezes de Sequeira, Diogo Henriques and Carlos Serrão  
Page 46

**ASSOCIATIVE PROGRAMMING AND MODELING: ABSTRACTIONS OVER COLLABORATION**  
Bent Bruun Kristensen  
Page 54

**A DECLARATIVE EXECUTABLE MODEL FOR OBJECT-BASED SYSTEMS BASED ON FUNCTIONAL DECOMPOSITION**  
Pierre Kelsen  
Page 63

**POSTERS**

**AVOIDING TWO-LEVEL SYSTEMS: USING A TEXTUAL ENVIRONMENT TO ADDRESS CROSS-CUTTING CONCERNS**  
David Greaves  
Page 71

# Software Engineering

**FULL PAPERS**

**DEVELOPING A CONFIGURATION MANAGEMENT MODEL FOR USE IN THE MEDICAL DEVICE INDUSTRY**  
Fergal McCaffery, Rory O'Connor and Gerry Coleman  
Page 81

**BRIDGING BETWEEN MIDDLEWARE SYSTEMS: OPTIMISATIONS USING DOWNLOADABLE CODE**  
Jan Newmarch  
Page 89

**ENGINEERING A COMPONENT LANGUAGE: COMPJAVA**  
Hans Albrecht Schmid and Marco Pfeifer  
Page 98
<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXPLORING FEASIBILITY OF SOFTWARE DEFECTS ORTHOGONAL CLASSIFICATION</td>
<td>107</td>
</tr>
<tr>
<td>Davide Falessi and Giovanni Cantone</td>
<td></td>
</tr>
<tr>
<td>MDE FOR BPM - A Systematic Review</td>
<td>118</td>
</tr>
<tr>
<td>Jose Manuel Perez, Francisco Ruiz and Mario Piattini</td>
<td></td>
</tr>
<tr>
<td>SHORT PAPERS</td>
<td></td>
</tr>
<tr>
<td>GENERIC FEATURE MODULES: TWO-STAGED PROGRAM CUSTOMIZATION</td>
<td>127</td>
</tr>
<tr>
<td>Sven Apel, Martin Kuhlemann and Thomas Leich</td>
<td></td>
</tr>
<tr>
<td>A FRAMEWORK FOR THE DEVELOPMENT OF MONITORING SYSTEMS SOFTWARE</td>
<td>133</td>
</tr>
<tr>
<td>Ildefonso Martinez-Martínez, Llanos Mora-López and Mariano Sidrach de Cardona</td>
<td></td>
</tr>
<tr>
<td>USING PRE-REQUIREMENTS TRACING TO INVESTIGATE REQUIREMENTS BASED ON</td>
<td>139</td>
</tr>
<tr>
<td>TACIT KNOWLEDGE</td>
<td></td>
</tr>
<tr>
<td>Andrew Stone and Pete Sawyer</td>
<td></td>
</tr>
<tr>
<td>USING LINGUISTIC PATTERNS FOR IMPROVING REQUIREMENTS SPECIFICATION</td>
<td>145</td>
</tr>
<tr>
<td>Carlos Videira, David Ferreira and Alberto Rodrigues da Silva</td>
<td></td>
</tr>
<tr>
<td>AN APPLICATION OF THE 5-S ACTIVITY THEORETIC REQUIREMENTS METHOD</td>
<td>151</td>
</tr>
<tr>
<td>Robert B. K. Brown, Peter Hyland and Ian C. Piper</td>
<td></td>
</tr>
<tr>
<td>LEARNING EFFECTIVE TEST DRIVEN DEVELOPMENT - Software Development Projects in an Energy Company</td>
<td>159</td>
</tr>
<tr>
<td>Wing Kam Amy Law</td>
<td></td>
</tr>
<tr>
<td>TOWARDS A LANGUAGE INDEPENDENT REFACTORING FRAMEWORK</td>
<td>165</td>
</tr>
<tr>
<td>Carlos López, Raúl Martirorona, Yania Crespo and Francisco Javier Pérez</td>
<td></td>
</tr>
<tr>
<td>A DYNAMIC ANALYSIS TOOL FOR EXTRACTING UML 2 SEQUENCE DIAGRAMS</td>
<td>171</td>
</tr>
<tr>
<td>Paolo Falcarin and Marco Torchiano</td>
<td></td>
</tr>
<tr>
<td>MINING ANOMALIES IN OBJECT-ORIENTED IMPLEMENTATIONS THROUGH EXECUTION TRACES</td>
<td>177</td>
</tr>
<tr>
<td>Paria Parsamanesh, Amir Abdollahi Foumani and Constantinos Constantinides</td>
<td></td>
</tr>
<tr>
<td>A DETECTION METHOD OF FEATURE INTERACTIONS FOR TELECOMMUNICATION SERVICES USING NEW EXECUTION MODEL</td>
<td>190</td>
</tr>
<tr>
<td>Sachiko Kawada, Masayuki Shimokura and Tadashi Obta</td>
<td></td>
</tr>
<tr>
<td>REACTIVE, DISTRIBUTED AND AUTONOMIC COMPUTING ASPECTS OF AS-TRM</td>
<td>196</td>
</tr>
<tr>
<td>E. Vasser, H. Kuang, O. Ormandieliu and J. Paquet</td>
<td></td>
</tr>
<tr>
<td>UNIFIED DESCRIPTION AND DISCOVERY OF P2P SERVICES</td>
<td>203</td>
</tr>
<tr>
<td>G. Athanasopoulos, A. Tsalgatidoum and M. Pantazoglou</td>
<td></td>
</tr>
<tr>
<td>BUILDING MAINTENANCE CHARTS AND EARLY WARNING ABOUT SCHEDULING PROBLEMS IN SOFTWARE PROJECTS</td>
<td>210</td>
</tr>
<tr>
<td>Sergiu Gordea and Markus Zanker</td>
<td></td>
</tr>
<tr>
<td>ADVANCES ON TESTING SAFETY-CRITICAL SOFTWARE - Goal-driven Approach, Prototype-tool and Comparative Evaluation</td>
<td>218</td>
</tr>
<tr>
<td>Guido Pennella, Christian Di Biagio, Gianfranco Pesce and Giovanni Cantone</td>
<td></td>
</tr>
<tr>
<td>A SYSTEMATIC REVIEW MEASUREMENT IN SOFTWARE ENGINEERING - State-of-the-art in Measures</td>
<td>224</td>
</tr>
<tr>
<td>Oswaldo Gómez, Hanna Oktaba, Mario Piattini and Félix Garcia</td>
<td></td>
</tr>
<tr>
<td>XV</td>
<td></td>
</tr>
</tbody>
</table>
TOWARDS ANCHORING SOFTWARE MEASURES ON ELEMENTS OF THE PROCESS MODEL
Bernhard Daubner, Bernhard Westfechtel and Andreas Henrich 232

WEB METRICS SELECTION THROUGH A PRACTITIONERS’ SURVEY
Julian Ruiz, Coral Calero and Mario Piattini 238

POSTERS

A PRIMITIVE EXECUTION MODEL FOR HETEROGENEOUS MODELING
Frédéric Boulanger and Guy Vidal-Naquet 247

INTRODUCTION TO CHARACTERIZATION OF MONITORS FOR TESTING SAFETY-CRITICAL SOFTWARE
Christian Di Biagio, Guido Pennella, Anna Lomartire and Giovanni Cantone 253

MODELLING THE UNEXPECTED BEHAVIOURS OF EMBEDDED SOFTWARE USING UML SEQUENCE DIAGRAMS
Hee-jin Lee, In-Gwon Song, Sang-Uk Jeon, Doo-Hwan Bae and Jang-Eui Hong 257

VIEWPOINT FOR MAINTAINING UML MODELS AGAINST APPLICATION CHANGES
Walter Cazzola, Ahmed Ghoneim and Gunter Saake 263

SYSML-BASED WEB ENGINEERING - A Successful Way to Design Web Applications
Haroon Tarawneh 269

REVERSE ENGINEERING ELECTRONIC SERVICES - From e-Forms to Knowledge
Costas Vassilakis, George Lapouras and Akrivi Katifori 273

A SCENARIO GENERATION METHOD USING A DIFFERENTIAL SCENARIO
Masayuki Makino and Atsushi Ohnishi 279

SYSTEM TEST CASES FROM USE CASES
Javier J. Gutiérrez, María J. Escalona, Manuel Mejías and Jesús Torres 283

DISTRIBUTED AND PARALLEL SYSTEMS

FULL PAPERS

ALGORITHMIC SKELETONS FOR BRANCH & BOUND
Michael Poldner and Herbert Kuchen 291

PARALLEL PROCESSING OF "GROUP-BY JOIN" QUERIES ON SHARED NOTHING MACHINES
Mohamad Al Hajj Hasian and Mostafa Bamba 301

IMPACT OF WRAPPED SYSTEM CALL MECHANISM ON COMMODITY PROCESSORS
Satoshi Yamada, Shigern Kasakabe and Hideo Taniguchi 308

SHORT PAPERS

A HYBRID TOPOLOGY ARCHITECTURE FOR P2P FILE SHARING SYSTEMS
Juan Pedro Muñoz-Gea, Josemaría Malgosa-Sanabria, Pilar Manzanares-Lopez, Juan Carlos Sanchez-Aarnoutse and Antonio M. Guirado-Puerta 319
AN APPROACH TO MULTI AGENT COOPERATIVE SCHEDULING IN THE SUPPLY CHAIN, WITH EXAMPLES
Joaquim Reis

TOWARDS A QUALITY MODEL FOR GRID PORTALS
Mª Ángeles Moraga, Coral Calero, Mario Piattini and David Walker

LANGUAGE-BASED SUPPORT FOR SERVICE ORIENTED ARCHITECTURES: FUTURE DIRECTIONS
Pablo Giambiagi, Olaf Owe, Gerardo Schneider and Anders P. Ravn

A METHODOLOGY FOR ADAPTIVE RESOLUTION OF NUMERICAL PROBLEMS ON HETEROGENEOUS HIERARCHICAL CLUSTERS
Wahid Nasri, Sonia Mahjoub and Slim Bouguerra

POSTERS

DEVELOPING A FAULT ONTOLOGY ENGINE FOR EVALUATION OF SERVICE-ORIENTED ARCHITECTURE USING A CASE STUDY SYSTEM
Binka Gwynne and Jie Xu

AN APPROACH TO MULTI-AGENT VISUAL COMPOSITION WITH MIXED STYLES
Joaquim Reis

A PEER-TO-PEER SEARCH IN DATA GRIDS BASED ON ANT COLONY OPTIMIZATION
Uroš Jovanović and Boštjan Slivnik

AUTHOR INDEX
ADAPTIVE INTEGRATION
OF ENTERPRISE AND B2B APPLICATIONS

Leszek A. Maciaszek
Department of Computing, Macquarie University, Sydney, NSW 2109, Australia
leszek@ics.mq.edu.au

Keywords: Application integration, software adaptiveness.

Abstract: Whether application integration is internal to the enterprise or takes the form of external Business-to-Business (B2B) automation, the main integration challenge is similar – how to ensure that the integration solution has the quality of adaptiveness (i.e. it is understandable, maintainable, and scalable)? This question is hard enough for stand-alone application developments, let alone integration developments in which the developers may have little control over participating applications. This paper identifies main strategic (architectural), tactical (engineering), and operational (managerial) imperatives for building adaptiveness into solutions resulting from integration projects.

1 INTRODUCTION

Today’s enterprise and e-business systems are rarely developed in-house from scratch. Most systems are the results of evolutionary maintenance of existing systems. Occasionally new systems are developed, but always with the intent to integrate with the existing software. New technologies emerge to facilitate development and integration of enterprise and e-business systems. The current thrust comes from the component technology standards and the related technology of Service Oriented Architecture (SOA).

This paper centers on conditions for developing adaptive complex enterprise and e-business systems. It concentrates on architectural design, engineering principles, and operational imperatives for developing such systems. An adaptive system has the ability to change to suit different conditions; the ability to continue into the future by meeting existing functional and nonfunctional requirements and by adjusting to accommodate any new and changing requirements. In some ways, an adaptive system is an antonym of a legacy system. A necessary condition of adaptiveness is the identification and minimization of dependencies in software. A software element A depends on an element B, if a change in B may necessitate a change in A.

Enterprise and e-business systems are complex – their properties and behavior cannot be fully explained by the understanding of their component parts. The software crisis has been looming at our doorsteps for years. Cobol legacy systems and the millennium bug are well known examples on the global scale, and the examples of individual software disasters are countless. Each time when faced with a crisis, we have been engaging the next technological gear to solve the problem. But also each time we have been introducing an additional level of complexity to the software and with it new and more troublesome non-adaptive solutions. The premise of this paper is that, unless we start producing adaptive systems, we are yet to face the first true software crisis.

2 DEVELOPMENT OR INTEGRATION?

Business has embraced the Internet-age technology with zeal. Thanks to application integration technologies, organizations can function as loosely connected networks of cooperating units. Development of stand-alone applications is all but history. Accordingly, the term “application development” is being replaced by the more accurate term – “integration development”.

Is-3
There are three integration levels (Erl, 2004). Figure 1 shows how the three levels are related to each other. All integration projects imply exchange of data between integrated applications. Typically this means that an application A gets access to application’s B database either directly or by data replication techniques.

![Integration Levels Diagram](image)

**Figure 1: Integration levels.**

At application level, application A uses the interfaces (services) of application B to request data or to request execution of the services provided by application B. A classic example is a Loan Broker application (Hohpe and Woolf, 2003), in which the integration solution negotiates loan terms with the banks for the customers. Although Loan Broker negotiates with many banks, the negotiations are separate for each bank. Hence, this is dyadic (point-to-point) integration.

Another successful example of dyadic integration is VMI (Vendor-Managed Inventory). In VMI integration, a vendor/supplier is responsible for monitoring and replenishing customer inventory at the appropriate time to maintain predefined levels. However, the ultimate goal of integration is the much more complex integration of detailed business processes performed by applications (and resulting in services and data production). At the process level new workflows of processes are designed that integrate processes already available in existing applications to provide a new value-added functionality.

Clearly, process level integration blurs the line between development and integration. At this level, an integration project is also a new development project. A new umbrella application is produced providing solutions that go beyond the sum of relationships between participating applications/enterprises and that go beyond simple integration effects.

The need for process-level integrations arises when businesses want to enter into collaborative environments to achieve joint outcomes, not just their own outcomes. Electronic marketplaces (e-markets) subscribe to that goal, but business factors limit e-market expansion. It is simply the case, that “sharing price or capacity information is often not advantageous to all parties in a supply chain or vertical market” (Christiaanse, 2005, p.97).

Where the business conditions are right, process-level integrations can flourish. Christiaanse (2005) provides two illustrative examples – transportation optimization and cash netting. Transportation optimization is a collaborative logistic application that consolidates various in-transit status messages for the trucks traveling around Europe so that empty trucks can be dynamically hired to take loads. Cash netting is designed to replace point-to-point invoice-payment processes by the “cash-netting” at the end of the day, i.e. once a day payment to/from each account, which is the value of shipments minus the value of receipts.

The main and overriding technology that drives integration development is SOA (service-oriented architecture) (Erl, 2004). SOA uses XML Web services as its implementation principle and introduces a new logical layer within the distributed computing platform. This new Integration layer defines a common point of integration across applications and across enterprises. In effect, SOA blurs the distinction between integration across applications and across enterprises. In effect, SOA blurs the distinction between integration and new distributed applications (because the reason for calling a service on the Integration layer is transparent to SOA – and the reason could be an integration or brand new application development).

Moreover, and not out of context, SOA blurs the distinction between a business process and a technology process (one no longer exclusively drives the other).

### 3 CLASSIFYING INTEGRATION

Integration projects are as much about the strategy as about the technology. As such, they have many dimensions and various mixing of these dimensions is needed to ensure the project’s business objectives and to choose the appropriate technology. Tables 1 and 2 provide two different two-dimensional viewpoints on integration projects.
Table 1: Two-dimensional view on integration projects.

<table>
<thead>
<tr>
<th>Dyadic integration</th>
<th>Hub integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal integration</td>
<td>External integration → SOA</td>
</tr>
<tr>
<td>Dyadic integration</td>
<td></td>
</tr>
<tr>
<td>File sharing</td>
<td>EDI</td>
</tr>
<tr>
<td>Remote procedures</td>
<td>Web services</td>
</tr>
<tr>
<td>Hub integration</td>
<td></td>
</tr>
<tr>
<td>Shared database</td>
<td>Message brokers</td>
</tr>
<tr>
<td>Workflows</td>
<td>Orchestration</td>
</tr>
</tbody>
</table>

Table 2: Another two-dimensional view on integration projects.

<table>
<thead>
<tr>
<th>Data integration</th>
<th>Process integration</th>
<th>Synchronous integration</th>
<th>Asynchronous integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data replication</td>
<td>Remote procedures</td>
<td>Data replication</td>
<td>File transfer</td>
</tr>
<tr>
<td>Portal sharing</td>
<td>Workflows</td>
<td>Portal sharing</td>
<td>Shared database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Messaging</td>
</tr>
</tbody>
</table>

The two criteria used in Table 1 are the integration business target and the number of integration participants. The business target can be the enterprise itself (internal integration) or another business (external B2B (business-to-business) integration) (cp. Linthicum, 2004). The number of participants can be just two parties in a point-to-point supply chain (dyadic integration) or more than two parties in a network structure (hub integration) (cp. Christiaanse, 2005).

The last column in Table 1 refers to external integration. Modern external integration solutions are based on SOA. A primitive forerunner of SOA as a technology for external integration has been EDI (Electronic Data Interchange) – a set of computer interchange standards for business documents. Web services are also defined by a collection of protocols and standards used for exchanging data between applications or enterprises. However, they make themselves available over the Internet for integration with applications and systems and they can be part of SOA infrastructure. Message brokers and orchestration engines are used for hub external integration. Message broker is a layer of software between applications integrated within the hub. It is a SOA component that ensures data transformation, merging and enrichment so that applications in the hub can communicate and collaborate.

Integration by means of remote procedures is based on an old piece of wisdom that data should be encapsulated by procedures. Accordingly, to access the data in another application, the client application invokes remotely appropriate procedures, which in turn supply the data. Two other internal integration solutions mentioned in Table 1 are shared database and workflows. Although shown as examples of hub integration, they are often used in simpler dyadic integrations. Also, when the business conditions are right, they can be used in external integration.

Any database is by definition shared, so talking about shared database emphasizes only the point that the database is used as an integration solution. Because a database can be shared by any number of users and applications, a shared database is an obvious vehicle for hub integration on the level of data.

The hub integration on the level of processes can be based on workflows. A workflow is a distributed business transaction governed by a process management function that ensures the integrated flow of execution of transactional tasks between many systems/applications.

The cells in both tables serve the purpose of providing examples of integration solutions. The simplest solutions referred to in Table 1 are file sharing and remote procedures. These two solutions are typically used in internal dyadic integration, but they are applicable in other more complicated integration projects as well.

Integration by means of file sharing means that files are transferred between applications. The integration effort concentrates merely on re-formatting the files to suit the receiving application.

Integration solutions listed in Table 1 can be analyzed from other angles, including the viewpoints taken in Table 2, namely degree of coupling and integration targets. In general, data integration is
more aligned with (better suited for) loosely coupled integration. This is because data can be easily put aside for later use. Conversely, process integration is more aligned with tightly coupled integration.

Data replication, portal sharing, and messaging are the three integration solutions listed in Table 2 but missing in Table 1. Data replication is classified as synchronous integration because replication servers of databases can be programmed to perform replications continuously and replicate data as the primary data is changing. Probably slightly controversially, portal sharing is classified in Table 2 as synchronous data integration. Portals are web sites or applications that provide access to a number of sources of information and facilities (portlets). They aggregate information from multiple sources into a single display. The display of information is synchronous but no any sophisticated process-level communication between portlets is normally assumed – hence, data integration.

Messaging is the primary technology for asynchronous process integration (Hohpe and Woolf, 2003). Based on the Publish/Subscribe model, messaging frameworks guarantee reliable delivery of messages in program-to-program communication while recognizing that synchronous communication with remote applications is difficult to achieve (yet asynchronous communication is frequently acceptable).

4 ASSURING ADAPTIVE INTEGRATION

Building adaptiveness into enterprise and e-business systems engages all three traditional levels of management – strategic, tactical and operational. From the system’s development perspective, the strategic level refers to the architectural solutions, the tactical level to the engineering decisions, and the operational level to the project controlling tasks. These three levels of management are used in the conventional top-down fashion when software is developed. We can say that system architecture defines adaptiveness, engineering activities deliver adaptiveness, and controlling tasks verify the existence of adaptiveness in an implemented system.

4.1 Defining Adaptiveness

A well known truth, unfortunately frequently forgotten in practice, is that the necessary condition for assuring adaptive integration (and any large software development for that matter) is that the integration adheres to strict and transparent architectural design. The architectural design itself must conform to a meta-architecture that is known to ensure the quality of adaptiveness in any compliant complex system. Meta-architecture determines the layers of the (necessary) hierarchical structure in a complex system and specifies allowed dependencies between and inside the layers.

There are many meta-architectures that in principle can support the quality of adaptiveness. However, most meta-architectures are undefined for that purpose. To be useful, a meta-architecture must classify possible dependencies according to their ripple effect, i.e. adverse chain reactions on client objects once a supplier object is modified in any way (Maciaszek, 2006). It must then offer guidelines, principles and patterns, which assist system developers in their quest to adhere to the architectural design while not “binding their hands and brains” too much (Maciaszek and Liong, 2005).

The pivotal meta-architecture, which we advocate, is called PCBMER. The PCBMER framework defines six hierarchical layers of software objects – Presentation, Controller, Bean, Mediator, Entity and Resource.

The PCBMER meta-architecture has evolved from earlier frameworks (e.g. Maciaszek and Liong, 2005) and has aimed at new development projects. We believe, however, that PCBMER can easily accommodate to integration projects. That belief is consistent with the earlier discussion that the demarcation line between development and integration is blurred and that any more sophisticated process-level integration is really a form of new application development. Nevertheless, PCBMER requires some extensions to account in the meta-architecture for the integration layer.

An important starting point for any extensions of PCBMER is that we can only ensure the quality of adaptiveness in the software that remains under our control. We can then only trust that parties that our software integrates with will also be adaptive. With this understanding in mind, we can distinguish between two meta-architectures that apply in integration projects.

The first integration meta-architecture applies to application-level integrations. This architecture subsumes also data-level integration. We call this...
architecture PCBMER-A (where A signifies an application-centric integration). Figure 2 shows a high-level view of the PCBMER-A meta-architecture.

The second integration meta-architecture applies to process-level integrations, providing support for inter-application and inter-organization communication. We call this architecture PCBMER-U (where U refers to a utility service that such meta-architectural solutions promise to deliver). Figure 3 is a high-level view of the PCBMER-U meta-architecture.

Figures 2 and 3 illustrate that the integration meta-architectures retain the Core PCBMER framework. Dependencies (dotted arrowed lines) between the core packages remain unchanged in integration projects. Hence, for example, Presentation depends on Controller and on Bean, and Controller depends on Bean. Note that the PCBMER hierarchy is not strictly linear and a higher-layer can have more than one adjacent layer below (and that adjacent layer may be an intra-leaf, i.e. it may have no layers below it).

The Bean package represents the data classes and value objects that are destined for rendering on user interface.
Unless entered by the user, the bean data is built from the entity objects (the Entity package). The Core PCBMER framework does not specify or endorse if access to Bean objects is via message passing or event processing as long as the Bean package does not depend on other packages.

The **Presentation** package represents the screen and UI objects on which the beans can be rendered. It is responsible for maintaining consistency in its presentation when the beans change. So, it depends on the Bean package. This dependency can be realized in one of two ways – by direct calls to methods (message passing) using the pull model or by event processing followed by message passing using the push model (or rather push-and-pull model).

The **Controller** package represents the application logic. Controller objects respond to the UI requests that originate from Presentation and that are results of user interactions with the system. In a programmable GUI client, UI requests may be menu or button selections. In a web browser client, UI requests appear as HTTP Get or Post requests.

The **Entity** package responds to Controller and Mediator. It contains classes representing “business objects”. They store (in the program’s memory) objects retrieved from the database or created in order to be stored in the database. Many entity classes are container classes.
The Mediator package establishes a channel of communication that mediates between Entity and Resource classes. This layer manages business transactions, enforces business rules, instantiates business objects in the Entity package, and in general manages the memory cache of the application. Architecturally, Mediator serves two main purposes. Firstly, to isolate the Entity and Resource packages so that changes in any one of them can be introduced independently. Secondly, to mediate between the Controller and Entity/Resource packages when Controller requests data but it does not know if the data has been loaded to memory or it is available in the database or it can be obtained from external sources.

The Resource package is responsible for all communications with external persistent data sources (databases, web services, etc.). This is where the connections to the database and SOA servers are established, queries to persistent data are constructed, and the database transactions are instigated.

For application-centric integration projects (Figure 2), the PCBMER-A meta-architecture enriches the Resource package with the RequiredInterfaces component. This component provides access to external applications. Although the component is called RequiredInterfaces, the access is not restricted to invoking Java-style interfaces implemented in collaborating applications. Any other integration levels are assumed and allowed, such as direct access to data, access to data encapsulated by accessor methods or by stored procedures, access to data rendered in portals, or access to web services.

For utility-centric integration projects (Figure 3), the PCBMER-U meta-architecture is explicitly extended with new “integration automation” components – Broker, Orchestration, and Service Registry. The first two implement the automation logic and depend on the utility’s application logic in Controller. Service Registry implements the “service discovery” and depends on the utility’s business logic in Mediator. All access to the integration layers of participating applications originates from either Mediator or Resource.

### 4.2 Delivering Adaptiveness

Once defined in a meta-architecture, an adaptive solution can be delivered through engineering work. It is the responsibility of engineers to ensure that architectural advantages are retained in the engineered product. The task is not easy because as always “the devil is in the detail”. To do the job, the engineers must be equipped with principles, patterns, implementation techniques, etc. that instrument the meta-architectural advantages and that explicitly address the adaptiveness criteria in the solution.

The Core PCBMER framework has a number of advantages resulting in minimization of dependencies. The main advantage is the separation of concerns between packages allowing modifications within one package without affecting the other (independent) packages or with a predictable and manageable effect on the other (dependable) packages. For example, the Presentation package that provides a Java application UI could be switched to a mobile phone interface and still use the existing implementation of Controller and Bean packages. That is, the same pair of Controller and Bean packages can support more than one Presentation UI at the same time.

The second important advantage is the elimination of cycles between dependency relationships and the resultant six-layer hierarchy with downward only dependencies. Cycles would degenerate a hierarchy into a network structure. Cycles are disallowed both between PCBMER packages and within each PCBMER package.

The third advantage is that the framework ensures a significant degree of stability. Higher layers depend on lower layers. Therefore, as long as the lower layers are stable (i.e. do not change significantly, in particular in interfaces), the changes to the higher layers are relatively painless. Recall also that lower layers can be extended with new functionality (as opposed to changes to existing functionality), and such extensions should not impact on the existing functionality of the higher layers.

The Core PCBMER meta-architecture enforces other properties and constraints that are not necessarily directly visible in Figures 2 and 3. Below is the list of the most important PCBMER engineering principles (cp. Maciaszek and Liong, 2005):

1. **Downward Dependency Principle (DDP)**
   - The DDP states that the main dependency structure is top-down. Objects in higher layers depend on objects in lower layers. Consequently, lower layers are more stable than higher layers. Interfaces, abstract classes, dominant classes and similar devices should encapsulate stable packages so that they can be extended when needed.

2. **Upward Notification Principle (UNP)**
   - The UNP promotes low coupling in a bottom-up communication between layers. This can be achieved by using asynchronous communication based on event processing. Objects in higher layers...
act as subscribers (observers) to state changes in lower layers. When an object (publisher) in a lower layer changes its state, it sends notifications to its subscribers. In response, subscribers can communicate with the publisher (now in the downward direction) so that their states are synchronized with the state of the publisher.

3. Neighbor Communication Principle (NCP)
   The NCP demands that a package can only communicate directly with its neighbor package as determined by direct dependencies between packages. This principle ensures that the system does not disintegrate to a network of intercommunicating objects. To enforce this principle, the message passing between non-neighboring objects uses delegation or forwarding (the former passes a reference to itself; the latter does not). In more complex scenarios, a special acquaintance package can be used to group interfaces to assist in collaboration that engages distant packages.

4. Explicit Association Principle (EAP)
   The EAP visibly documents permitted message passing between classes. This principle recommends that associations are established on all directly collaborating classes. Provided the design conforms to PCBMER, the downward dependencies between classes (as per DDP) are legitimized by corresponding associations. Associations resulting from DDP are unidirectional (otherwise they would create circular dependencies). It must be remembered, however, that not all associations between classes are due to message passing. For example, both-directional associations may be needed to implement referential integrity between classes in the entity package.

5. Cycle Elimination Principle (CEP)
   The CEP ensures that circular dependencies between layers, packages and classes within packages are resolved. Circular dependencies violate the separation of concerns guideline and are the main obstacle to reusability. Cycles can be resolved by placing offending classes in a new package created specifically for that purpose or by forcing one of the communication paths in the cycle to communicate via an interface.

6. Class Naming Principle (CNP)
   The CNP makes it possible to recognize in the class name the package to which the class belongs. To this aim, each class name is prefixed in PCBMER with the first letter of the package name (e.g. EInvoice is a class in the Entity package). The same principle applies to interfaces. Each interface name is prefixed with two capital letters – the first is the letter “I” (signifying that this is an interface) and the second letter identifies the package (e.g. ICInvoice is an interface in the Controller package).

7. Acquaintance Package Principle (APP)
   The APP is the consequence of the NCP. The acquaintance package consists of interfaces that an object passes, instead of concrete objects, in arguments to method calls. The interfaces can be implemented in any PCBMER package. This effectively allows communication between non-neighboring packages while centralizing dependency management to a single acquaintance package.

4.3 Verifying Adaptiveness
   The PCBMER meta-architecture together with the seven principles defines a desired model to produce adaptive systems. However, the meta-architecture is a theoretical objective which may or may not be fulfilled in practice. Also, it is possible to have multiple designs (and corresponding implementations), all of which conforming to the meta-architecture, yet exhibiting various levels of “goodness”. What we need is to be able to measure how “good” particular software solution is and whether or not it conforms to the meta-architecture. The overall task is called the roundtrip architectural modelling in Maciaszek (2005).
   Therefore, to verify adaptiveness in an integration solution, we need to define structural complexity metrics able to compute cumulative dependencies between the solution’s implementation objects. The dependencies can be defined on messages, events, classes, components, etc. (Maciaszek and Liong, 2003).
   From the perspective of a system architect and maintainer, the dependencies between classes provide the most valuable metric of system complexity and adaptiveness. It is, therefore, important to make these dependencies explicit and to uncover any hidden dependencies. The Cumulative Class Dependency (CCD) is a measure of the total number of class dependencies in a system.
   DEFINITION: Cumulative Class Dependency (CCD) is the total “adaptiveness” cost over all classes $C_i$ in a system of the number of classes $C_j$ to be potentially changed in order to modify each class $C_i$. 

IS-10
The CCD definition is intentionally simple. In particular it does not, by itself, judge the quality of the design. Its value is in comparisons between two or more designs for the same system. To this aim, the CCD computation strives to validate if a particular design conforms to a chosen meta-architecture (such as PCBMER). Uncovering a class dependency that invalidates the architectural framework leads to the only sensible assumption that the required dependency structure in the system is broken. This in turn means that any dependency is possible and the system adaptiveness has eluded management controls.

The calculation of CCD for a particular design starts by assuming the adherence to the architectural framework. If the framework is found to be broken, the CCD is calculated as if each class depended on any other class in the system. Such worst-scenario CCD can be computed using a probability theory method called combinations counting rule. It computes the number of different combinations of pairs of dependent classes which can be formed from the total number of classes in the design. With the above in mind, the generic cumulative class dependency formula for the Core PCBMER is shown in the equation below (this is a generic formula and other formulas may apply to specific PCBMER architectures derived from the Core framework). The formula assumes that access to packages is encapsulated by hub objects (Maciaszek, 2006). These could be Java-style interfaces, dominant classes, and similar devices, which force single channels of communication between packages.

\[
\text{hubPCBMER} \text{CCD} = \sum_{i=1}^{\text{root}} o_i (o_i - 1) / 2 + \sum_{j=1}^{\text{root}} p_{j+1}
\]

where:
- \( o \) is the number of objects in each package \( i \), including any hub objects,
- \( p_{j+1} \) is the number of objects in each directly adjacent package above any leave package minus any hub object (this computes the number of potential downward paths to all hub objects in the adjacent packages),
- \( \text{hubPCBMER} \text{CCD} \) is a cumulative class dependency in a hub hierarchy representing the PCBMER meta-architecture.

Note that the formula accommodates the fact that the PCBMER framework permits a lower-layer package to be communicated from more than one higher-layer package. These higher-layer packages are considered to be “directly adjacent”, hence the formula applies as stands. Note that because only downward dependencies are allowed, the communication from higher-layer packages retains the hierarchical properties of the PCBMER framework.

The CCD equation ensures polynomial growth of dependencies between architectural layers represented as packages, while allowing exponential growth of class dependencies within layers. However, the exponential growth can be controlled by grouping classes within a layer into nested packages (as packages can contain other packages). The communication between nested packages can then be performed using hubs.

Measuring adaptiveness of designs and programs cannot be done manually. Maciaszek and Liong (2003) describes a tool, called DQ (Design Quantifier), which is able to analyse any Java program, establish its conformance with a chosen adaptive meta-architecture, compute complete set of dependency metrics, and visualize the computed values in UML class diagrams.

Although not supported by DQ, tools like DQ should be able to visualize dependencies by producing call graphs. Ideally, a call graph could be a variant of a UML sequence diagram. A call graph can be used for the change impact analysis and to answer “what-if” questions such as “which methods are affected if a particular method is modified?”

5 SUMMARY

The purpose and all-overriding importance of achieving the quality of adaptiveness in software is in ensuring that the software becomes a long-lasting business asset, not just business cost. This observation is particularly true for integration projects, which by definition tend to deliver software with greater competitive advantages.

In Maciaszek (2006) and elsewhere, we explained the interplay between software complexity and adaptiveness, showed that hierarchical structures with hubs minimize complexity, talked about lessons from studying structure and behaviour of living systems, provided classifications of object dependencies, and introduced the PCBMER meta-architecture.

In this paper, we extended earlier work related to new application developments to the software integration projects. We argued that the demarcation line between new development and integration is
blurry and that the similar strategies and principles of software production apply. In particular, the Core PCBMER meta-architecture can be successfully adapted to integration projects and we showed necessary architectural extensions. We addressed software engineering practices and technologies that could guarantee the compliance of an implemented software system with the PCBMER meta-architecture and its principles. Finally, we talked about reverse-engineering verification procedures to substantiate in metrics the level of compliance of an integration solution with the adaptivity criteria.

REFERENCES


BRIEF BIOGRAPHY

Leszek A. Maciaszek is an Associate Professor of Computing at Macquarie University in Sydney, Australia. He obtained his MSc and PhD degrees in Informatics from University of Economics, Wroclaw, Poland (in 1972 and 1977, respectively). He has been working interchangeably in academia and industry. His assignments have included national organizations, international corporations and educational institutions in countries spanning four continents, including Australia, France, Germany, Italy, Kuwait, Macao, Malaysia, Poland, Singapore, Thailand, The Netherlands, and USA. Leszek's main areas of expertise evolve around the modeling, design, implementation and integration of enterprise information systems. He has authored about 120 publications related to databases, object technology, software engineering, systems modeling, and workgroup computing. Leszek's research interests in defining architectural, engineering and organizational imperatives for supportable enterprise systems stem from the experience gained in numerous consultancies, in particular as a project leader and software architect. Leszek has authored and co-authored a number of textbooks and reference books. His main books are: "Database Design and Implementation" (Prentice Hall, 1990), "Requirements Analysis and System Design" (Addison Wesley, 2001; translated to Chinese, Italian and Russian; 2nd edition published in 2005), and "Practical Software Engineering. A Case-Study Approach" (Addison Wesley, 2005; co-authored with Bruc Lee Liong).
Keywords: Ambient Intelligence, Sensor Networks, Artificial Intelligence.

Abstract: Ambient Intelligence is a multi-disciplinary approach which aims to enhance the way environments and people interact so as to make the places we live and work in more beneficial. Smart Homes is one example of such systems but the idea can be also used in relation to hospitals, public transport, a factory and other environments. The achievement of Ambient Intelligence largely depends on the technology deployed (sensors and devices interconnected through networks) as well as on the intelligence of the software used for decision-making. We review the characteristics of systems with Ambient Intelligence, provide examples of applications and highlight possible future developments in the area.

1 SUMMARY

Ambient Intelligence (IST Advisory Group, 2001) is growing fast as a multi-disciplinary approach which can allow many areas of research to have a real beneficial influence in our society. The basic idea behind Ambient Intelligence is that by enriching an environment with technology (mainly sensors and devices interconnected through a network), a system can be built such that based on the real-time information gathered and the historical data accumulated, decisions can be taken to benefit the users of that environment.

An example of such environment is a Smart Home, i.e., a house equipped to bring advanced services to its users, see, e.g., (Augusto and Nugent, 2006). Several artifacts and items in a house can be enriched with sensors to gather information about their use and in some cases even to act independently without human intervention. Some examples of such devices are electrodomestics (e.g., cooker and fridge), household items (e.g., taps, bed and sofa) and temperature handling devices (e.g., air conditioning and radiators). Expected benefits of this technology can be: (a) increasing safety (e.g., by monitoring lifestyle patterns or the latest activities and providing assistance when a possibly hazardous situation is developing), (b) comfort (e.g., by adjusting temperature automatically, etc.), and (c) economy (e.g., controlling the use of lights).

Recent applications include the use of Smart Homes to provide a safe environment where people with special needs can have a better quality of life. For example in the case of people at early stages of senile dementia (the most frequent case being elderly people suffering from Alzheimer’s disease) the system can be tailored to minimize risks and ensure appropriate care at critical times by monitoring activities, diagnosing interesting situations and advising the carer. There are already many ongoing academic research projects with well established Smart Homes research labs in this area, see for example Domus (Pigot et al., 2002), Aware Home (Abowd et al., 2002), MavHome (Cook, 2006), and Gator Tech Smart Home (Helal et al., 2005).

Other applications are also feasible and relevant and the use of sensors and smart devices can benefit:

1. Health-related applications. Hospitals can increase the efficiency of their services by monitoring patients’ health and progress by monitoring activities in their rooms and they can also increase safety by, for example, only allowing authorized personnel and patients to have access to specific areas and devices.

2. Public transportation sector. Public transportation sector can benefit from extra technology including satellite services, GPS-based spatial location, vehicle identification, image processing and other technologies to make transport more fluent and hence more efficient and possibly reduce accidents.

3. Education services. Education-related institutions can use technology to track students progression on their tasks, frequency of attendance to specific places and health related issues like advising on their diet regarding their habits and the class of in-takes they opted for.

4. Emergency services. Safety-related services like fire brigades can improve the reaction to a hazard.
by locating the place more efficiently and also by preparing the way to reach the place in connection with street services. The prison service can also quickly locate a place where a hazard is occurring or is likely to occur and prepare a better access to it for the security personnel.

5. Production-oriented places. Production-centred places like factories can self-organize according to the production/demand ratio of the goods produced. This will demand careful correlation between the collection of data through sensors within the different sections of the production line and the pool of demands via a diagnostic system which can advise the people in charge of the system at a decision-making level.

Well-known leading companies have already invested heavily in the area. For example Philips (Philips, 2006) has already developed Smart Homes for the market including innovative technology on interactive displays. Siemens (Siemens, 2006) has invested in Smart Homes and in factory automation. Nokia (Nokia, 2006) has been also leading developments in Smart Homes and the industry of communications. VTT (VTT, 2006) has developed systems which advise inhabitants of Smart Homes on how to modify their daily behaviour to improve their health.

This context creates a good research niche many existing areas of Computer Science can relate to. There is a distributed system consisting of a hardware platform with numerous devices interconnected through a network. They take and produce significant amounts of complex information most of which once gathered has to be stored in databases. The decision-making core of the system needs to be flexible and capable to anticipate situations even in the presence of uncertainty. Such complex systems poses significant challenges from a Software and Knowledge Engineering point of view. Within all this rich and complex technological setting users interact permanently with the system in complex ways performing their daily tasks and is up to the system to assist them as non-intrusively as possible.

In this invited talk we will review the characteristics of these systems, consider examples of applications and highlight the challenges and areas for potential development from the perspective of researchers and professionals attending ICSOFT2006.

REFERENCES


BRIEF BIOGRAPHY

Dr. Augusto’s research experience since 1998 has been focused on AI-related problems and mostly related to temporal reasoning. His latest research activities have explored the application of spatio-temporal reasoning to Ambient Intelligence in general and Smart Homes in particular. He is currently editing a book on Artificial Intelligence Techniques applied to Smart Homes to be published by Springer Verlag as a volume in the Lecture Notes on Artificial Intelligence series during 2006. During his 12 years of academic work Dr. Augusto contributed with 21 journal, edited volumes, and book chapter publications, more than 30 conference and workshop publications, 4 presentations in the form of invited talks, tutorials and seminars for international conferences and companies and 5 extend seminars invited by different Universities. Some of his latest editorial work includes the book “Designing Smart Homes: the role of Artificial Intelligence”. He is currently a co-chair of the 2nd Workshop on AI Techniques for Ambient Intelligence (AITAmI2007), co-located with IJCAI’07, and recently has been a co-chair of the 4th International Conference on Smart Homes and Telecare (ICOST’2006) held in Belfast. He has previously served as Steering/Program Committee member for more than ten international conferences and co-chaired an international workshop (MSVVEIS) for the last four editions.
Abstract: ‘Scales of measure’ are fundamental to a specification method we have developed called Planguage. They are central to the definition of all scalar attributes; that is, to all the performance (especially quality attributes) and resource attributes. You can learn the art of developing your own tailored scales of measure for the performance and resource attributes, which are important to your organization or system. You cannot rely on being ‘given the answer’ about how to quantify. You will lose control over your current vital system performance concerns if you cannot or do not quantify the critical attributes.

1 FINDING AND DEVELOPING SCALES OF MEASURE AND METERS

The basic advice for identifying and developing scales of measure and meters (practical methods for measuring) for scalar attributes is as follows:

1. Try to re-use previously defined Scales and Meters. Examples [Posm. www].
2. Try to modify previously defined Scales and Meters.
3. If no existing Scale or Meter can be reused or modified, use common sense to develop innovative home-grown quantification ideas.
4. Whatever Scale or Meter you start off with, you must be prepared to learn. Obtain and use early feedback, from colleagues and from field tests, to redefine and improve your Scales and Meters.

2 REFERENCE LIBRARY FOR SCALES OF MEASURE

‘Reuse’ is an important concept for, sharing experience and saving time when developing Scales. You need to build reference libraries of your ‘standard’ scales of measure. Remember to maintain details supporting each ‘standard’ Scale, such as Source, Owner, Status and Version (Date). If the name of a Scale’s designer is also kept, you can probably contact them for assistance and ideas. Here is a template for keeping reusable scales of measure.

| Tag: | <assign a tag name to this Scale>. |
| Version: | <date of the latest version or change>. |
| Owner: | <role/email of who is responsible for updates/changes>. |
| Status: | <Draft, SQC Exited, Approved>. |
| Scale: | <specify the Scale with defined [qualifiers]>.
| Alternative Scales: | <reference by tag or define other Scales of interest as alternatives and supplements>.
| Qualifier Definitions: | <define the scale qualifiers, like ‘for defined [Staff]’, list their options, like {Nurse, Doctor, Orderly}>.
| Meter Options: | <suggest Meter(s) appropriate to the Scale>. |
| Known Usage: | <reference projects & specifications where this Scale was actually used in practice with designers’ names>.
| Known Problems: | <list known or perceived problems with this Scale>.
| Limitations: | <list known or perceived limitations with this Scale>.

Example: This is a draft template, with <hints>, for specification of scales of measure in a reference library. Many of the terms used here are defined in Competitive Engineering [www & CE]. See example below for sample use of this template.
Tag: Ease of Access.
Owner: Rating Model Project (Bill).
Scale: Speed for a defined [Employee Type] with defined [Experience] to get a defined [Client Type] operating successfully from the moment of a decision to use the application.
Alternative Scales: None known yet.
Qualifier Definitions:
  - Employee Type: {Credit Analyst, Investment Banker, …}.
  - Experience: {Never, Occasional, Frequent, Recent}.
  - Client Type: {Major, Frequent, Minor, Infrequent}.
Meter Options:
  - Test all frequent combinations of qualifiers at least twice. Measure speed for the combinations.
Known Problems: None recorded yet.
Limitations: None recorded yet.

Example of a ‘Scale’ specification for a Scale reference library. This exploits the template in the previous example.

3 REFERENCE LIBRARY FOR METERS

Another important standards library to maintain is a library of ‘Meters.’ Meters support scales of measure by providing practical methods for actually measuring the numeric Scale values. ‘Off the shelf’ Meters from standard reference libraries can save time and effort since they are already developed and are more or less ‘tried and tested’ in the field.

It is natural to reference suggested Meters within definitions of specific scales of measure (as in the template and example above). Scales and Meters belong intimately together.

4 MANAGING ‘WHAT’ YOU MEASURE

It is a well-known paradigm that you can manage what you can measure. If you want to achieve something in practice, then quantification, and later measurement, are essential first steps for making sure you get it. If you do not make critical performance attributes measurable, then it is likely to be less motivating for people to find ways to deliver necessary performance levels. They have no clear targets to work towards, and there are no precise criteria for judgment of failure or success.

5 PRACTICAL EXAMPLE: SCALE DEFINITION

‘User-friendly’ is a popular term. Can you specify a scale of measure for it?

Here is my advice on how to tackle developing a definition for this.

1. If we assume there is no ‘off-the-shelf’ definition that could be used (there are [POSEM], [CE]):
   - Be more specific about the various aspects of the quality. There are many distinct dimensions of qualities such as usability, maintainability, security, adaptability and their like [CE]. List about 5 to 15 aspects of some selected quality that is critical to your project.
   - For this example, let’s select ‘environmentally friendly’ as the one of many aspects that we are interested in, and we shall work on this below as an example.

2. Invent and specify a Tag: ‘Environmentally Friendly’ is sufficiently descriptive. Ideally, it could be shorter, but it is very descriptive left as it is. We indicate a ‘formally defined concept’ by capitalizing the tag.

   [Tag: Environmentally Friendly.]

Note, we usually don’t explicitly specify ‘Tag: ’ but
this sometimes makes the tag identity clearer.

3. Check there is an Ambition statement, which briefly describes the level of requirement ambition. ‘Ambition’ is a defined Planguage parameter. More parameters follow, below.

| Ambition: A high degree of protection, compared to competitors, over the short-term and the long-term, in near and remote environments for health and safety of living things. |

4. Ensure there is general agreement by all the involved parties with the Ambition definition. If not, ask for suggestions for modifications or additions to it. Here is a simple improvement to my initial Ambition statement. It actually introduces a ‘constraint’.

| Ambition: A high degree of protection, compared to competitors, over the short-term and the long-term, in near and remote environments for health and safety of living things, which does not reduce the protection already present in nature. |

5. Using the Ambition description, define an initial ‘Scale’ (of measure) that is somehow quantifiable (meaning – you can meaningfully attach a number to it). Consider ‘what will be sensed by the stakeholders’ if the level of quality changes. What would be a ‘visible effect’ if the quality improved? My initial, unfinished attempt, at finding a suitable ‘constraint’.

| Scale: The % change in positive (good environment) or negative directions for defined [Environmental Changes]. |

My first Scale parameter draft, with a single scalar variable.

However, I was not happy with it, so I made a second attempt. I refined the Scale by expanding it to include the ideas of specific things being effected in specific places over given times:

| Scale: % destruction or reduction of defined [Thing] in defined [Place] during a defined [Time Period] as caused by defined [Environmental Changes]. |

This is the second Scalar definition draft with four scalar variables. These will be more-specifically defined whenever the Scale is applied in requirement statements such as ‘Goal’.

This felt better. In practice, I have added more [qualifiers] into the Scale, to indicate the variables that must be defined by specific things, places and time periods whenever the Scale is used.

6. Determine if the term needs to be defined with several different scales of measure, or whether one like this, with general parameters, will do. Has the Ambition been adequately captured? To determine what’s best, you should list some of the possible sub-components of the term (that is, what can it be broken down into, in detail?). For example:

| Thing: [Air, Water, Plant, Animal]; |
| Place: [Personal, Home, Community, Planet]. |
| Thing: = [Air, Water, Plant, Animal]. |
| Place: Consists of [Personal, Home, Community, Planet]. |

Definition examples of the scale qualifiers used in the examples above. The first example means: ‘Thing’ is defined as the set of things Air, Water, Plan and Animal (which, since they are all four capitalized, are themselves defined elsewhere). Instead of just the colon after the tag, the more explicit Planguage parameter ‘Consists Of’ or ‘=’ can be used to make this notation more immediately intelligible to novices in reading Planguage.

Then consider whether your defined Scale enables the performance levels for these sub-components to be expressed. You may have overlooked an opportunity, and may want to add one or more qualifiers to that Scale. For example, we could potentially add the scale qualifiers ‘… under defined [Environmental Conditions] in defined [Countries]…’ to make the scale definition even more explicit and more general.

Scale qualifiers (like … ‘defined [Place]’…) have the following advantages:

- they add clarity to the specifications
- they make the Scales themselves more reusable in other projects
- they make the Scale more useful in this project: specific benchmarks, targets and constraints can be specified for any interesting combination of scale variables (such as, ‘Thing = Air’).

7. Start working on a ‘Meter’ – a specification of how we intend to test or measure the performance of a real system with respect to the defined Scale. Remember, you should first check there is not a standard or company reference library Meter that you could use. Try to imagine a practical way to measure things along the Scale, or at least sketch one out. My example is only an initial rough sketch.
This Meter specification is a sketch defined by a set of three rough measurement concepts. These at least suggest something about the quality and costs with such a measuring process. The ‘Meter’ must always explicitly address a particular ‘Scale’ specification.

The Meter will help confirm your choice of Scale as it will provide evidence that practical measurements can feasibly be obtained on a given Scale of measure.

8. Now try out the Scale specification by trying to use it for specifying some useful levels on the scale. Define some reference points from the past (Benchmarks) and some future requirements (Targets and Constraints). For example:

Environmentally Friendly:

Ambition: A high degree of protection, compared to competitors, over the short-term and the long-term, in near and remote environments for health and safety of living things, which does not reduce the protection already present in nature.

Scale: % destruction or reduction of defined [Thing] in defined [Place] during a defined [Time Period] as caused by defined [Environmental Changes].

Past [Time Period = Next Two Years, Place = Local House, Thing = Water]: 20% <- intuitive guess.
Record [Last Year, Cabin Well, Thing = Water]: 0% <- declared reference point.
Trend [Ten to Twenty Years From Now, Local, Thing = Water]: 30% <- intuitive. "Things seem to be getting worse."

Fail [End Next Year, Thing = Water, Place = Eritrea]: 0%. "Not get worse."

Wish [Thing = Water, Time = Next Decade, Place = Africa]: <3% <- Pan African Council Policy.
Goal [Time = After Five Years, Place = <our local community>, Thing = Water]: <5%.
Environmentally Friendly:

Ambition: A high degree of protection, compared to competitors, over the short-term and the long-term, in near and remote environments for health and safety of living things, which does not reduce the protection already present in nature.

--- Some scales of measure candidates – they can be used as a complimentary set ---

- Air: Scale: % of days annually when <air> is <fit for all humans to breath>.
- Water: Scale: % change in water pollution degree as defined by UN Standard 1026.
- Earth: Scale: Grams per kilo of toxic content.
- Predators: Scale: Average number of <free-roaming predators> per square km, per day.
- Animals: Scale: % reduction of any defined [Living Creature] who has a defined [Area] as their natural habitat.

Many different scales can be candidates to reflect changes in a single critical factor.

Environmentally Friendly is now defined as a ‘Complex Attribute,’ because it consists of a number of ‘elementary’ attributes: {Air, Water, Earth, Predators, Animals}. A different scale of measure now defines each of these elementary attributes. Using these Scales we can add corresponding Meters, benchmarks, (like past) constraints (like Fail) and target levels (like Goal) to describe exactly how Environmentally Friendly we want to be.

Level of Specification Detail

How much detail you need to specify, depends on what you want control over, and how much effort it is worth. The basic paradigm of Planguage is you should only elect to do what pays off for you. You should not build a more detailed specification than is meaningful in terms of your project and economic environment. Planguage tries to give you sufficient power of articulation to control both complex and simple problems. You need to scale up, or down, as appropriate. This is done through common sense, intuition, experience and organizational standards (reflecting experience). But, if in doubt, go into more detail. History says we have tended in the past to specify too little detail about requirements. The result consequently has often been to lose control, which costs a lot more than the extra investment in requirement specification.

6 LANGUAGE CORE: SCALE DEFINITION

This section discusses the specification of Scales with qualifiers.

The Central Role of a ‘Scale’ within Scalar Attribute Definition. The specified Scale of an elementary scalar attribute is used (re-used!) within all the scalar parameter specifications of the attribute (that is, within all the benchmarks, the constraints and the targets). In other words, a Scale parameter specification is the heart of a specification. Scale is essential to support all the related scalar level parameters: for example Past, Record, Trend, Goal, Budget, Stretch, Wish, Fail and Survival.

Each time a different scalar level parameter is specified, the Scale specification dictates what has to be defined numerically and in terms of Scale Qualifiers (like ‘Staff = Nurse’). And then later, each time a scalar level parameter definition is read, the Scale specification itself has to be referenced to ‘interpret’ the meaning of the corresponding scale level specification. So the Scale is truly central to a scalar definition. For example ‘Goal [Staff = Nurse] 23%’ only has meaning in the context of the corresponding scale: for example ‘Scale: % of defined [Staff] attending the operation’. Well-defined scales of measure are well worth the small investment to define them, to refine them, and to re-use them.

Specifying Scales using Qualifiers. The scalar attributes (performance and resource) are best measured in terms of specific times, places and events. If we fail to do this, they lose meaning. People wrongly guess other times, places and events than you intend, and cannot relate their experiences and knowledge to your numbers. If we don't get more specific by using qualifiers, then performance and resource continues to be a vague concept, and there is ambiguity (which times? which places? which events?).

Further, it is important that the set of different performance and resource levels for different specific time, places and events are identified. It is likely that the levels of the performance and resource requirements will differ across the system depending on such things as time, location, role and system component.
Decomposing complex performance and resource ideas, and finding market-segmenting qualifiers for differing target levels is a key method of competing for business.

Here is some more detail about subjects shown above as examples.

Embedded Qualifiers within a Scale. A Scale specification can set up useful ‘scale qualifiers’ by declaring embedded scale qualifiers, using the format ‘defined [<qualifier>]’. It can also declare default qualifier values that apply by default if not overridden, ‘defined [<qualifier>: default: <User-defined Variable or numeric value>]’. For example, […]default: Novice].

Additional Qualifiers. However, embedded qualifiers should not stop you adding any other useful additional qualifiers later, as needed, during scale related specification (such as Goal or Meter). But, if you do find you are adding the same type of parameters in almost all related specifications, then you might as well design the Scale to include those qualifiers. A Scale should be built to ensure that it forces the user to define the critical information needed to understand and control a critical performance or resource attribute. This implies that scale qualifiers serve as a check list of good practice in defining scalar level specifications such as Past and Goal.

Here is an example of how locally defined qualifiers (example in a Goal specification) can make a quality specification more specific. In this example we are going to see how a requirement can be conditional upon an event. If the event is not true, the requirement does not apply.

Now we apply those definitions below:

<table>
<thead>
<tr>
<th>Quality A:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Quality Requirement.</td>
</tr>
<tr>
<td>Scale: % by value of Goods delivered that are returned for repair or replacement by consumers.</td>
</tr>
<tr>
<td>Meter [Development]: Weekly samples of 10,</td>
</tr>
<tr>
<td>[Acceptance]: 30 day sampling at 10% of representative cases,</td>
</tr>
<tr>
<td>[Maintenance]: Daily sample of largest cost case,</td>
</tr>
<tr>
<td>Fail [European Union, Assumption A]: 40% &lt;- European Economic Members.</td>
</tr>
<tr>
<td>Goal [EU and EEU members, Positive Trade Balance]: 50% &lt;- EU Trade.</td>
</tr>
</tbody>
</table>

Some of the user-defined terms used here (like EU Trade) are more fully defined in the example above this one.

The Fail and the Goal requirements are now defined partly with the help of qualifiers. The Goal to achieve 50% (or more, is implied) is only a valid plan if ‘Positive Trade Balance’ is true. The Fail level requirement of 40% (or worse, less, is implied) is only valid if ‘Assumption A’ is true. All qualifier conditions must be true for the level to be valid.

7 PRINCIPLES: SCALE SPECIFICATION

1. The Principle of ‘Defining a Scale of Measure’
   If you can’t define a scale of measure, then the goal is out of control.
   Specifying any critical variable starts with defining its units of measure.

2. The Principle of ‘Quantification being Mandatory for Control’
   If you can’t quantify it, you can’t control it.¹
   If you cannot put numbers on your critical system variables, then you cannot expect to communicate about them, or to control them.

3. The Principle of ‘Scales should control the Stakeholder Requirements’
   Don’t choose the easy Scale, choose the powerful Scale.

¹ Paraphrasing a well known old saying.
Select scales of measure that give you the most direct control over the critical stakeholder requirements. Chose the Scales that lead to useful results.

4. The Principle of ‘Copycats Cumulate Wisdom’
Don’t reinvent Scales anew each time – store the wisdom of other Scales for reuse.

Most scales of measure you will need, will be found somewhere in the literature, or can be adapted from existing literature.

5. The Cartesian Principle
Divide and conquer said René – put complexity at bay.

Most high-level performance attributes need decomposition into the list of sub-attributes that we are actually referring to. This makes it much easier to define complex concepts, like ‘Usability’, or ‘Adaptability,’ measurably.

6. The Principle of ‘Quantification is not Measurement’
You don’t have to measure in order to quantify!

There is an essential distinction between quantification and measurement.

\[
\text{Be clear about one thing. Quantification is not the same as Estimation and Measurement.}
\]

“I want to take a trip to the moon in nine picoseconds” is a clear requirement specification without measurement.”

The well-known problems of measuring systems accurately are no excuse for avoiding quantification – Quantification allows us to communicate about how good scalar attributes are or can be – before we have any need to measure them in the new systems.

7. The Principle of 'Meters Matter'
Measurement methods give real world feedback about our ideas.

A ‘Meter’ definition determines the quality and cost of measurement on a scale; it needs to be sufficient for control and for our purse.

8. The Principle of 'Horses for Courses’
Different measuring processes will be necessary for different points in time, different events, and different places.


Exact numbers are ambiguous unless the units of measure are well-defined and agreed.

Formally-defined scales of measure avoid ambiguity. If you don’t define scales of measure well, the requirement level might just as well be an arbitrary number.

10. The Principle of ‘Being Sure About Results’
If you want to be sure of delivering the critical result – then quantify the requirement.

Critical requirements can hurt you if they go wrong – and you can always find a useful way to quantify the notion of ‘going right; to help you avoid doing so.

8 CONCLUSIONS
This paper has tried to show the pragmatic detail available in Planguage for specification of performance scales of measure; and for exploiting those scales of measure to define benchmarks, targets and constraints. There is in fact much more language facility available in Planguage as it is defined in the Competitive Engineering text to express concepts surrounding quantified quality and other performance requirements and analytical specifications. We hope this sample itself was useful to the reader, and that they are tempted to take the trouble to access more of the language [CE].

REFERENCES


2 ‘Horses’ for Courses is UK expression indicating something must be appropriate for use, fit for purpose.
3 There is no universal static scale of measure. You need to tailor them to make them useful.
(Usability) and Chapter 19 Software Engineering Templates.


[RE] *Requirements Engineering* (about 400 Slides) considerable examples and theory.

**BRIEF BIOGRAPHY**


Tom Gilb was born in Pasadena CA in 1940. He moved to England in 1956, then two years later he joined IBM in Norway. Since 1963, he has been an independent consultant and author. For further information and papers, see www.Gilb.com. His new book Competitive Engineering goes into more detail on quantification of quality.

QUANTIFYING STAKEHOLDER VALUES

Tom Gilb
Iver Holtersvei 2, NO-1410 Kolbotn, Norway
Tom@Gilb.com, www.Gilb.com, +47 66801697

Abstract: Here are some questions we need to ask about stakeholder value. How can we determine the overall value of a system? How is this value related to the performance characteristics of the system? How can we engineer the value to meet stakeholder expectations? How can we test and measure the real value? Can we contract for system payment by value, or do we have to restrict ourselves to payment for performance levels? Is there any way to quantify the overall value of a system as a function of a set of system attributes?

The performance-to-value relation.

- It is intuitively obvious that, as system performance attributes vary, the values of that system, to defined stakeholders, at defined times, under defined circumstances, vary.
- It is equally obvious that there are levels of performance so low that they give no value at all – or even make all other value attributes worthless (imagine zero availability); and increases in performance that give little or no improvement in value (imagine 99.999999999% availability).
- One central problem is that many engineers have not learned to quantify some performance characteristics, particularly some quality characteristics (for example usability, adaptability). They are indeed quantifiable but are treated ‘qualitatively’ with words (‘very user-friendly’, ‘highly flexible’) in most cases.
- There is no strong tradition for such attributes of being quantified (as there is with reliability and availability for example).
- This quality quantification problem must be confronted if we are going to be able to compute corresponding stakeholder value for those quality variations.

Here is my suggestion for the fundamental principles of stakeholder value quantification.

1. Stakeholder (implied hereafter) Value depends on a set of factors.

   - 1.1 Value depends on the level of single (performance, function, constraint, and cost) attributes.
   - 1.2 Value also depends on a defined set of such system attributes existing simultaneously.
   - 1.3 Value depends on a point in time synchronizing with a set of external circumstances (example markets, laws, transportation costs)

2. We can usefully distinguish between estimated value, calculated value, contractual value, potential value, realized value, and perceived value.

3. The systems engineering effort must consciously manage the necessary system characteristics, in order to deliver the minimum conditions (the system characteristics) for allowing a stakeholder to potentially derive the potential value.

4. The systems engineering effort cannot be responsible for achieving necessary stakeholder value conditions that are outside their control (like a ‘willing market’). But, they should be responsible for identifying such outside conditions, for making stakeholders aware of such external conditions, and for designing the system so that the stakeholder has every opportunity to take advantage of, or deal with, the external conditions.

5. All binary conditions (like ‘legal’) can be specified in a testable manner
6. All scalar (variable) attributes that determine value can be specified quantitatively (in particular, all qualities), can be designed into the system consciously, and measured and tested as to the level actually attained. In particular all value-critical system qualities can be defined quantitatively. See the defined process in [Gilb, 2005], Ch 5 for quantification.

7. If the system characteristics necessary for delivering stakeholder value are clarified and quantified, then the basis for ‘no cure no pay’ contracting is laid. Or at least strong motivation to deliver the necessary levels of system performance on time, is present.

Here are some practical tools for managing stakeholder value:

**Value Policy:**

Here are some specific policy ideas that somehow need to become part of your corporate systems engineering culture.

1. The main purpose of our systems engineering work is to enable our stakeholders to get maximum value for cost.

2. We will systematically identify and specify all relevant stakeholders, and all values we can influence for them, and will make a cost-effective effort to deliver the system attributes that will enable our stakeholders to derive maximum value for them.

3. We will develop our systems engineering culture, training, motivation and tools so as to make value delivery happen in practice.

**Planguage**

The Planning language (‘Planguage’) detailed in my book ‘Competitive Engineering’ (CE, (Gilb, 2005)) has dozens of practical specification tools for enabling us to analyze both system value-drivers, and value analysis and planning. Some of these tools will be hinted at in the rest of the paper (example, templates).

Qualifiers: qualifiers are Planguage specification tools that allow us to be explicit about the necessary times, events and spaces that are a prerequisite for defining when, where, and if, value can be created.

For example:

- **Usability:**
  - **Stakeholders:** Local Olympic Committee.
  - **Scale:** time to learn a typical task.
  - **Goal** [China, Teenagers, At Olympics, If Our product is Purchased] less than 10 minutes.
  - **Impacts** [If Training Paid for by Local Olympic Committee] Stadium Personnel Costs.

- The product quality requirement (named ‘Usability’) Goal (the level we require to get value) has a set of ‘qualifier conditions’ that must all be ‘true’ or valid, for the ‘less than 10 minutes’ Goal level to be a valid requirement.

- The Goal level is designed to increase the value of the system we engineer, by reducing the ‘Stadium Personnel Costs’ for the Stakeholder ‘Local Olympic Committee’.

**Templates**

- **Performance Template**
  - A simple template for specifying a system performance characteristic, might correspond to the example above:

- **<name tag>**
  - **Stakeholders:** <name the top few stakeholders who are impacted by this particular requirement>.

- **Scale:** <define a scale of measure for this performance attribute>.

- **Goal** [<when?>, <where>, <event>]: <specify the necessary, but practical and economic performance level>.

- **Impacts:** <specify the stakeholder value specification that the Goal level is intended to impact>.

- The <hints in fuzzy brackets> are digitally erasable help for giving the intended information about the requirement.
A much more detailed template is published in my book [CE] at the end of each corresponding chapter.

**Impact Estimation Tables (IET)**

- IETs are for relating system attributes (as ‘means’) to Value Objectives.
- Impact Estimation Tables are P-language tools that
  - Help us analyze and present the relationships between any useful set of technical system objectives, and any set of stakeholder values that these technical objectives are intended to impact.
  - The IE Tables can be constructed so that the basis for the estimates is clear, even though the connection may be weakly based, risky, or even more well-founded.
- The Impact Estimation table is a useful way of summarizing the degree of formal planning done thus far for satisfying stakeholder values. But even when the IE Table shows a strong plan in place, it needs to be confirmed step by step through Evolutionary step measurement. It is not as reliable a tool as actual measurement of practical experience with real stakeholders.
- Here is an Impact Estimation Table framework for analyzing stakeholder values that are affected by engineered system performance attributes, such as system qualities.
- Impact Estimation Table skeleton: this table format can be used to help us analyze the projected impacts, of engineered system characteristics, on a set of stakeholder values.

- The estimates can be rough or refined depending on circumstances and ability. SA1 -> V1 means that here is where we would place an estimate of the impact of System Attribute SA1 (example Usability) on Stakeholder Value V1 (example Staff Costs). We would use the ‘percentage language’ to indicate our estimate. 100% means that the Attribute Goal level, if attained in practice, would serve to meet 100% of the Stakeholder Value Goal level. 0% would indicate there was no expected impact at that intersection. A ‘?’ would indicate that we did not understand the relationship, and indicate a risk until further study was done.
- The ‘Sum %’ horizontally would be a rough indication of the degree to which planned system attribute levels would satisfy the stakeholder value.
- The Sum/Cost estimate would be an indication of the efficiency of a given system performance attribute level in satisfying a set of stakeholder values.

An impact estimate can be a subjective or consensus number for the sake of discussion. But an impact estimate can also have the following additional disciplines added in order to make it more useful.

- The ‘Real impact’ number on a defined scale (like 6.0 ‘minutes to learn a task’) can be estimated
- An uncertainty can be estimated based on the known range of experience, or a best-case/worst-case stipulation (50%±20% for example)
- A source of the estimate can be documented (‘<- The Times Jan 17 2006, p 23, James Whittaker’)

<table>
<thead>
<tr>
<th>System Attributes ---&gt;</th>
<th>SA1</th>
<th>SA2</th>
<th>SA3</th>
<th>SA4</th>
<th>SA5</th>
<th>Sum %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stakeholder Values (Vn)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V1:</td>
<td>SA1-&gt;V1</td>
<td>SA2-&gt;V1</td>
<td>100%</td>
<td>0%</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>V2:</td>
<td>SA3-&gt;V2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V3:</td>
<td>SA2-&gt;V3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum</td>
<td>ΣSA1-n</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sum/Cost</td>
<td>Σ/€</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The facts, or ‘evidence’ for the estimation, the ‘estimate credibility’ basis, can be cited (‘1500 IT Systems investigated in UK in 2005’)

The credibility level of the estimate can be specified, based on a scale from 0.0 (no credibility) to 1.0 (perfect credibility).

• The table is not intended to arrive at a final truth. The IET is intended to help us display our level of understanding in a systematic way, so that the next steps can be taken in a systematic way. For example, do some practical trials on all high-benefit but low-credibility estimates, first.

Summary
• Stakeholder values can be expressed quantitatively. Stakeholder values can be satisfied by an engineered system, through specific system attribute levels (such as qualities).
• It is possible to analyze the relationship between our systems engineering efforts, and the projected degree of stakeholder satisfaction.
• The stakeholder values themselves must be described quantitatively.
• The impacts of our system strategies, mostly the system performance characteristics, on specific stakeholder values, can be estimated.
• We can systematically manage the probable satisfaction of stakeholder values, by something better than verbal and qualitative communication.

REFERENCES


This book goes into considerable practical detail about how to specify and analyze stakeholder values and system requirements for impacting those stakeholder values. It contains 393 occurrences of ‘stakeholder’, 304 occurrences of ‘value’.


BRIEF BIOGRAPHY

Tom has been an independent consultant, teacher and author, since 1960. He mainly works with multinational clients; helping improve their organizations, and their systems engineering methods.


Other books are ‘Software Inspection’ (with Dorothy Graham, 1993), and ‘Principles of Software Engineering Management’ (1988). His ‘Software Metrics’ book (1976, OoP) has been cited as the initial foundation of what is now CMMI Level 4.

Tom’s key interests include business metrics, evolutionary delivery, and further development of his planning language, ‘Planguage’. He is a member of INCOSE and is an active member of the Norwegian chapter NORSEC. He participates in the INCOSE Requirements Working Group, and the Risk Management Group.
Keywords: Metamodeling, design, semantic integration, ontologies.

Abstract: This paper strives for demonstrating “metamodels in action” which means showing concrete applications of this concept. Based on a literature survey we develop a taxonomy that helps classifying existing application scenarios concerning the dimensions of domain, design, and integration and briefly describe some of the existing work we came across. Furthermore, we provide an insight into the area of semantic integration and how metamodels can be brought together with ontologies in this context. The paper is concluded with an outlook on important future work in the field of metamodeling.

1 INTRODUCTION

Information systems design and implementation is a complex task that usually implies the use of a variety of different approaches such as requirements engineering (Castro et al., 2002) or conceptual modeling (Wand and Weber, 2002) whereas also social psychological aspects have to be considered (Filipe, 2002).

This paper is focused to provide a brief overview on the use of metamodels in the area of information systems and other computer science disciplines. In order to do so, we chose the empirical method of a literature survey. Alas, current state-of-the-art of search technologies like renown search engines on the Internet are still not able to process requests like “How are metamodels applied in the software engineering community?” producing results with sufficient recall and precision values\(^1\). Therefore, we had to chose a different approach that shall be briefly described in the following.

We examined the Internet archives of 18 renown journals in different communities of computer science that is to say software engineering, databases, knowledge engineering, and information systems. These high quality journals are published by IEEE, ACM, Springer, Elsevier, or IOSPress – a full list can be found in appendix A in section 5.

When using the provided search facilities of the journals an interesting observation was made. The idea of enhancing computer systems with a better understanding of semantics has been in the spotlight for some years now, just think of the initiative in the context of the Semantic Web (Berners-Lee et al., 2001). One of the goals of this initiative is to improve search results in the way to really provide us with the information we need. In order to do so the use of ontology-like constructs (cf. (Garshol, 2004), (Obrst, 2003)) shall help us to cope with different writings, synonyms, homonyms, and the like. But reality is still different. It is still impossible to use “metamodeling” (American English) as search string also getting hits with the keyword “metamodelling” (British English) and vice versa. Different ways of writing like “meta-modeling”, “meta-modeling”, and “meta modeling” are still resulting in different search results. There is also a long way to go until we are really able to obtain semantically related hits when searching for “metamodeling” that might for example be indexed with “conceptual modeling”.

In the end we used “metamodeling, “meta-modeling, “meta modeling” each written once in American and once in British English as well as, “metamodel”, “meta-model”, and “meta model” as nine different search strings. The actual inquiry was

---

\(^1\)Recall is calculated as the ratio of the number of documents retrieved that are relevant compared to the total number of documents that are relevant. Precision on the other hand is the number of documents retrieved that are relevant divided by the total number of documents that are retrieved. (Witten and Frank, 2005)
accomplished in July 2006 and provided us with a corpus of 77 articles dealing with our subject of interest whereas no search restrictions concerning the time the papers have been published have been applied. This corpus was used as basis for the further analysis of classification possibilities for metamodel applications.

It has to be stressed here that we concentrated on the tasks that can be handled with metamodels which means investigating their practical use to solve real-world problems. We are not talking about a classification of how metamodels can actually be represented. If we would have concentrated on this design issue we would have spoken about the application of logical rules or object-orientation and their reproduction in computer systems. We also have not taken into account the actual practical implementation of the identified usages which would have implied dealing with service-oriented architectures, databases and the like (see figure 1).

The remainder of the paper is organized as follows: Section 2 gives a brief overview on modeling and metamodeling in order to ensure a common understanding and to stress the difference between linguistic and ontological metamodeling. Thereafter, in section 3 our classification taxonomy of metamodel applications is developed as well as some of the existing work from literature is described. Section 4 provides an insight into the importance of semantic integration and the benefits of combining the concept of metamodeling with ontologies in this context. The paper is concluded in section 5 also giving an outlook on important future work.

2 MODELING AND METAMODELING

Basically, in the area of computer science models are seen as "a representation of either reality or vision." ((Whitten et al., 2004), p. 187) Therefore, they describe things either as they are or as they should be.

Of course, this representation is not able to include all aspects of the original but can only focus on some of them (property of reduction) and a model is always intended for a specific purpose (property of pragmatics). (Stachowiak, 1973)

Models can be classified according to the language that is used for their creation. Non-linguistic or iconic models use signs and symbols that have an apparent similarity to the concepts of the real world that are being modeled. Linguistic models on the other hand use basic primitives (i.e. signs, characters, numbers, ...) that do not contain any apparent relationship to the part of reality being modeled except the one that is defined in an explicit way. (Strahlinger, 1996) Nearly all models used in computer science are of the latter linguistic type2 on which we restrain ourselves hereafter.

The next step before we can talk about metamodels is to clarify how models are actually built. Here the notion of a modeling technique comes into play which describes the modeling constructs of a modeling language (usually entities, relationships and attributes) and a modeling procedure that defines how these constructs have to be combined in order to create a valid model (see figure 2). Following Harel and Rumpe a modeling language now consists of syntax which focuses “purely on notational aspects” and semantics which defines the meaning (Harel and Rumpe, 2000). Kühn extends this view in that he separates notation from syntax as he defines notation as the “representation of the elements of the language” (Kühn, 2004). Syntax then is how the representation elements are allowed to be combined. We consider this distinction as important as it allows for changing only syntax or only notation in the context of method engineering (Henderson-Sellers, 2006) without affecting the other.

Metamodels most generally are defined as “models of (other similar) models”. A graphical representation similar to figure 3 is typically used to explain this. On the bottom layer 0 there is reality with all its objects and facts that shall be modeled. This is done

---

2Linguistic models can be further distinguished in being realized with textual and graphical/diagrammatic languages (Harel and Rumpe, 2000).
with the help of a modeling language. For instance, when creating a database for let’s say the management of student data we can use the Entity-Relationship modeling technique (ERM, (Chen, 1976)) in order to abstract reality. The available modeling primitives of ERM (i.e. entities, weak entities, relationships with different cardinalities) are described in the metamodel on layer 2 using a meta modeling language. This modeling primitives can be defined by another meta layer, layer 3, which is called meta-meta-layer or meta²-layer containing a meta²-model using a meta²-modeling language. Thus, here the concept of metamodeling is used as a means of language definition. Atkinson and Kühne denote this as linguistic metamodeling. (Atkinson and Kühne, 2003)

But according to these authors this “traditional” point of view on metamodeling covers only one of two important dimensions. It does not explicitly consider that there is not only a linguistic instantiation of concepts like student is an instance of entity, for instance, but also an ontological one as student is a person, too. In linguistic metamodeling student and person are situated on the same layer, whereas from an ontological point of view person would be on a meta layer. This aspect is called ontological metamodeling whereas Atkinson and Kühne emphasize that sophisticated metamodeling environments should give equal importance to both identified metamodeling dimensions.

Now that the basics of models and metamodeling have been recapitulated it is time to proceed to our approach for a metamodeling taxonomy.

3 A TAXONOMY OF METAMODELING APPROACHES

First we tried to classify metamodel applications according to the discipline of computer science in which they were used. So we tried to distinguish metamodeling in the database-, software engineering, knowledge engineering-, and information systems community. However, it was soon clear that this would not be a proper distinction dimension as we saw that metamodels were used to solve the same type of problems in different disciplines. Therefore, the second approach was to take usages of metamodeling as single classification criteria. Here we used categories like “design template”, “method engineering”, “processing of data”, or “integration”: Again, this proved to be too narrow-minded as it turned out that some of these categories are depending on each other, there are functional dependencies, for instance, between design and method engineering.

In the end we decided to use a kind of combination of the two classification approaches described above resulting in a three-dimensional classification space (see figure 4).

The dimension that can be described most easily is the one of the domain in which the metamodeling concept is applied (the y-axis in our classification space). The actual values of this dimension are quite similar to the above mentioned disciplines but not congruent. From the study of our document corpus (cf. section 1) we derived the following values: “data processing”, “knowledge representation”, “requirements engineering”, “information systems”, “business process- & workflow management”, “decision support”, and “business”. It should be stressed at this point that another document corpus which is different from the one we used most likely will lead to other dimension values. This might be true for the following two classification dimensions as well but to a much smaller extent.

The bottom of our classification space (i.e. mathematically speaking the x- and z-axis) is built by the two most basic dimensions of tasks that can be solved with the help of metamodels: design and integration. We will now generally discuss what these dimensions mean and give concrete examples in the following subsections.

Design as first task involves both the prescriptive definition of not yet existing as well as the descriptive modeling of already existing “objects” of interest. As
will be shown later we distinguish between macro-level and micro-level design.

Integration as last dimension now denotes the application of metamodeling for bringing together different existing “artefacts” of potentially various kinds that have been generated using different metamodels. This can now, for instance, mean the integration of heterogeneous data sources or the mapping between model instances (layer 1 in figure 3). It is important to notice that in order to be able to integrate artefacts that are already described by a metamodel, at least a meta²-model is needed. Thus, whereas design can be realized with only one meta-layer we need a meta²-layer for integration (see figure 5). Of course, this also implies that we cannot handle integration tasks without having properly defined all involved metamodels on the design level.

Now that we have explained the dimensions of our taxonomy for metamodels, some concrete applications that we have come across during our literature survey shall be presented in the following.

3.1 Design

Considering design two different aspects have to be distinguished: On the one hand the metamodeling concept with its abstraction layers (see figure 3) can be used to realize a kind of inheritance mechanism. We call this “micro-level design” as the inner structure of data models, representation languages, or the like is defined here. “Macro-level design” on the other hand generates concrete metamodels (layer 2 in figure 3) that act as templates or reference structures that can be used to deal with a variety of tasks. The macro-perspective metamodels contain a lot of application specific semantics and their implementation is therefore restricted to particular domains while the micro-perspective use is generic.

3.1.1 Macro-Level Design

One widespread use of metamodels is top-down as design templates or reference frameworks for certain tasks within a specified domain. The advantage of this use is quite apparent: a commonly accepted understanding of relevant “real-life”-concepts is guaranteed and new model instances can be created in a structured way whereas it is ensured that all relevant aspects are taken into account and nothing of importance is forgotten.

In our literature study we found a wide variety of applications of macro-level design metamodels (see figure 6). In the area of data processing, formal metamodels are, for instance, used to describe basic ETL (extraction-transformation-loading) tasks in the context of the extraction, processing and insertion of operational data in “cleaned” databases of data warehouses. The identified metamodel constructs are hereby used to generate templates (e.g. “domain mismatch”, “fact table”, ...) that can be lined up to compose complex ETL processes. ((Vassiliadis et al., 2001), (Vassiliadis et al., 2005))

Design templates are especially used in the information systems community. Some concrete applications include templates for the implementation of web-based systems (Nikolaidou and Anagnostopoulos, 2005), an agent-oriented metamodel for organizations and information systems (Wagner, 2003), and federated information systems (Jarke et al., 1997).

(Rolland et al., 1995) provide a general metamodel for business processes and (Chiu et al., 1999) define a metamodel for adaptive workflow management systems. As workflow management systems use strict predefined process control structures, they are not that
suited for supporting knowledge intensive processes that usually need much more flexibility. For the support of knowledge workers that have to act in such a flexible environment a new type of systems have been proposed: case handling systems that assist rather than strictly guide an user. (Aalst et al., 2005) introduce a metamodel for the cases that are provided by such systems.

Another macro-level design application can be found in (Rosemann and Green, 2002) where a metamodel of the Bunge-Wand-Weber model (BWWM) is generated which is basically a model for defining requirements when designing and realizing information systems. The original BWWM defines five fundamental and about 30 other constructs (see (Kayed and Colomb, 2005)) and it is argued to be hard to understand because of this complexity. Rosemann and Green see a variety of advantages in their metamodel of the BWWM: it clarifies the understanding, simplifies the communication, is a means for structuring and analyzing, and can finally be used to derive new modeling techniques.

We also came across two metamodels that are directly intended for business use. (Herbst, 1996) gives a metamodel for the definition of business rules, and (Krishna et al., 2004) for the creation of eContracts.

### 3.1.2 Micro-Level Design

To recapitulate, micro-level design is concerned with the definition of the structure of data models or representation languages. This ability is often used in the context of data processing for enhancing the reflection mechanisms\(^4\) of database management systems. To be more specific, the TIGUKAT object model (Peters and Ozsu, 1993) uses the concept of meta- and meta\(^2\)-objects in order to define types with specific behavior. This behavior is then passed to the objects of the instance level. In this context metamodeling is used to realize an inheritance mechanism that is known from the area of object-oriented programming, for instance. The FORM data model (Kim and Park, 1997) for representing heterogeneous types of entities and relationships in an organization does a quite similar thing as it uses metamodeling “to express meta-knowledge that allows a system to enforce generic patterns of object behavior rather than a number of specific actions”. A recent work proposes a framework for uniform representation of and access to data models, schema, and data. This “uni-level description” (ULD) also makes use of the classic metamodeling layers with one conceptual difference: schema and data are each stored on the same metalayer as compared to earlier approaches that put these

---

\(4\)Reflection is the ability of a system to manage information about itself and to reason about this information. (Peters and Ozsu, 1993)
In the areas of information systems and software engineering integration is often used in the context of method engineering. (Hillegersberg and Kumar, 1999), for instance, build one integrated metamodel of object-oriented methodologies concerning all development phases (i.e., analysis, design, and programming). The advantage of this approach is that this single metamodel can also be used to generate object-oriented program code (OOPC) from analysis/design model instances (OOADM) as a mapping between OOADM and OOPC can be established on the metamodel-layer.

Situational method engineering which is “concerned with the tuning of methods and techniques to the specific characteristics of a certain project” (Hofstede and Verhoef, 1997) is an integration application that can be found quite often in literature including the following articles: (Brinkkemper et al., 1999), (Dominguez and Zapata, ress), (Hofstede and Verhoef, 1997), (Beydoun et al., 2005) and (Prakash, 1999).

Finally, we came across an application of integration in the requirements engineering domain. (Nissen and Jarke, 1999) explicitly mention the use of a meta²-model to create an integrated view that also enables the computer-based support of team- and goal-oriented analysis methods like the informal method JAD. They also make use of the model transformation capabilities of the meta²-approach that “come for free” in that different requirements models that have been realized in unequal modeling languages (therefore using various metamodels) can be mapped.

It can be seen that metamodels are an adequate means for integration. But so far the described approaches are not able to realize semantic integration which “is concerned with the use of explicit semantic descriptions” (Kalfoglou et al., 2005) most often provided in the form of ontologies (Alexiev et al., 2005). The next section will show how metamodels and ontologies can be combined.

4 SEMANTIC INTEGRATION USING METAMODELS

The topic of integrating data and ensuring the interoperability of information systems is of great practical importance which can already be seen by the fact that according to Gartner up to 40% of the companies’ information technology budgets are spent on integration issues (Haller et al., 2005). The heterogeneities that have to be dealt with in this context are usually classified to be of syntactical, structural or semantical nature ((Alexiev et al., 2005), (Obrst, 2003)) whereas resolving the latter seems to be most laborious as 60-80% of the resources of integration projects are spent on reconciling semantic heterogeneities (Doan et al., 2004).

In order to be able to overcome the heterogeneities of resources – regardless if they are data, information systems, or anything else – they have to be represented in an adequate way. For this task linguistic, diagrammatic languages (cf. section 2) are often well suited like demonstrated by UML or ERM. These languages together with the concept of metamodeling are able to express syntactical and structural aspects as well as what we would like to denote as type semantics. This type semantics is defined through the process of linguistic metamodeling and allows reasoning such as, for instance, student is derived from the metamodel construct entity and therefore is a kind of real world object and not a relationship. But in this context we are not able to state anything about the semantics of student itself. It can by no means be reasoned that this term denotes a human person that can be male or female and who is attending an university-like institution. We would like to call this information inherent semantics as it describes a kind of “inner meaning” of modeled resources that is exceeding the type semantics that is being inherited by the elements of the metamodel-layer.

This inherent semantics can now be made explicit by linking model elements representing resources with concepts of ontologies, a process that is called lifting (Kappel et al., 2006) or also ontology anchoring (Brinkkemper et al., 1999) which is the quintessence of semantic integration. Lifting reflects in our opinion what (Atkinson and Kühne, 2003) denote as ontological metamodeling and is of course not limited to the model-layer but can be applied to the meta- or meta²-layer as well. This shall be illustrated in figure 8 in the following example.

Imagine two models that have been created using different metamodels and are now to be integrated which means that semantically related model elements have to be found. The “classical” metamodel-based approach would follow the path of linguistic instantiation which means that, for instance, it would be reasoned that meta-classes A and B are related because they are derived from the same meta²-class Ω (parent classes in figure 8 are given in brackets). Then the next step would be to say that model elements a and b belong together as they are instances of A and B. To be more specific we could assume A and B to be meta-classes of performance figures (ontology construct I) and the task would be to sum up all monetary figures (ontology construct II) of models 1 and 2. Ontology construct III now could stand for quantity figures. We see that a is a quantity figure then and

---

3The models in this figure are not created using a specific modeling language and therefore the notation is arbitrary without any implicit semantics.
should therefore not be added up to $b$. This conclusion can not be drawn with the information provided by the linguistic metamodeling process but only because of the additional information originating from the lifting of the model constructs.

Of course, one could argue now that the distinction between monetary and quantity figures could have been realized on the metamodel-layer as well. Basically, this is true but not really preferable because of a very specific feature of ontologies: They are by definition commonly accepted within communities as they reflect a shared and sometimes even standardized conceptualization\footnote{(Guarino, 1998) denotes this type of ontologies as reference ontologies.} compared to metamodels that are often only valid for specific tools or organizations. The advance of lifting (meta-)model concepts to ontology concepts is therefore founded in a reduced mapping complexity. If $n$ metamodels are to be mapped with each other the complexity of $n \times (n - 1)/2 \approx O(n^2)$ in the case of bidirectional point-to-point mappings can be reduced to $O(n)$ with one intermediate ontology.

Applying ontologies for semantic markup also allows for making use of all research results in the field of ontology mapping – see (Kalfoglou and Schorlemmer, 2005) or (Noy, 2004) for introducing surveys.

Recapitulatory, we believe that the combination of metamodeling and ontologies provides excellent means to solve the task of extensive integration handling all syntactical, structural and semantic heterogeneity. Some related work in this context can be found in (Kramler et al., 2006) and (Kappel et al., 2006) who deal with model transformations in the area of software engineering. (Roser and Bauer, 2005) also utilizes the idea of ontology-based transformation but like in the aforementioned papers the use of lifting is restricted to the metamodel layer. An approach that makes use of lifting on all (meta-)model layers can be found in (Terrasse et al., 2006).

5 CONCLUSION AND OUTLOOK

In this paper we described the results of a literature survey that aimed for describing “metamodels in action”. We presented a taxonomy for classifying concrete application scenarios of metamodeling according to domain, design, and integration and described some of existing work we came across.

Furthermore, we delivered an insight into the important field of semantic integration showing how metamodels can be enriched with ontology concepts. We are convinced that this approach will greatly enhance integration and interoperability both on the conceptual (EMI) as well as on the technical level (EAI).

\textit{Method engineering} for the combination of modeling paradigms is another important metamodel application scenario which will bring together descriptive, decision support-, and predicative models. Model-
driven business engineering will help for managing the interdependencies of corporations’ elements.

Further research work in the area of metamodels can focus on one of these identified applications. Another option is to elaborate on the two other aspects of metamodeling identified in figure 1: How metamodels are actually realized or how the identified tasks are implemented in practice.

APPENDIX A

A total of 18 journals has been surveyed whereas 11 delivered search results according to our defined nine search strings (see section 1). The surveyed journals are as follows [number of relevant retrieved documents is given in brackets]:

- Data & Knowledge Engineering [6]
- IEEE Transactions on Knowledge and Data Engineering (T-KDE) [2]
- ACM Transactions on Information Systems (TOIS; Formerly: ACM Transactions on Office Information Systems) [3]
- Communications of the Association for Information Systems (CAIS) [0]
- Electronic Journal of Information Systems Evaluation (EJISE) [0]
- Information Systems [21]
- Information Systems Research [4]
- Journal of Intelligent Information Systems [0]
- Journal of the Association for Information Systems (JAIS) [0]
- ACM SIGSOFT Software Engineering Notes [10 selected out of 87]
- ACM Transactions on Software Engineering and Methodology (TOSEM) [15]
- Empirical Software Engineering [0]
- IEE Proceedings (sic!) - Software Engineering [0]
- IEEE Transactions on Software Engineering (T-SE) [7]
- VLDB Journal, The - The International Journal on Very Large Databases [0]
- Distributed and Parallel Databases [1]
- ACM Transactions on Database Systems (TODS) [5]

REFERENCES


BRIEF BIOGRAPHY

Prof. Karagiannis studied Computer Science at the Technical University of Berlin and graduated several visit stays in the USA and Japan. From 1987 until 1992 he was business unit manager for Business Information Systems at the Research Institute for Applied Knowledge Management (FAW) in Ulm. 1993 he founded the Department of Knowledge Engineering at the Insitute for Computer Science and Business Informatics at the University of Vienna, focusing on Knowledge Management, Business Intelligence and Meta-Modelling. Prof. Karagiannis has published lot of scientifical research papers in the field of Databases, Expert Systems, Business Process Management, Workflow-Systems and Knowledge Management. He is the author of two books concerned with Knowledge Databases and Knowledge Management and is engaged in national and EU-funded research projects. The Business Process Management Approach he established, which is concerned with the thematic of Knowledge- and Business Process Management, has been succesfully implemented in sev-eral service companies. He founded the european software- and consulting company BOC ITC Ltd. (http://www.boc-eu.com), which realised the development and implementation of the business process management toolkit ADONIS.
Abstract: The FAME project uses method engineering to construct a methodological approach for agent-oriented software development. Its precursor was a project utilizing the object-oriented OPEN Process Framework, in which its repository of OO-focussed method fragments was extended to support various agent-oriented methodological approaches. In this talk, I will show how method engineering provides an excellent base for constructing situation specific software engineering methodologies for both object and agent software development. Both OPF and FAME use an existing repository coupled to an appropriate metamodel (which in the near future will be the new ISO standard metamodel ISO24744, itself based on the concept of powertypes). This flexible, yet standardized repository supplies method fragments that are then configured to support specific projects. In addition, all existing, and new, OO and AO methodologies can be recreated, thus providing an industry strength resource for object-oriented and agent-oriented software development.

BRIEF BIOGRAPHY

Brian Henderson-Sellers is Director of the Centre for Object Technology Applications and Research and Professor of Information Systems at the University of Technology, Sydney (UTS). He is author or editor of 27 books and is well-known for his work in OO and AO methodologies (MOSES, COMMA, OPEN, OOSPICE, FAME), OO metrics and metamodelling. More recently, he has chaired workshops at OOPSLA and AOIS on agent-oriented methodologies. He is Editor of the International Journal of Agent-Oriented Software Engineering and on the editorial board of Journal of Object Technology, Software and Systems Modelling and International Journal of Cognitive Informatics and Natural Intelligence. In July 2001, Professor Henderson-Sellers was awarded a Doctor of Science (DSc) from the University of London for his research contributions in object-oriented methodologies.
ARCHITECTURAL STYLES IN SERVICE ORIENTED DESIGN

Marten J. van Sinderen
University of Twente
The Netherlands

Abstract: Service oriented architecture (SOA) is characterized by the separation of (service) functionality from (service) provider. A service denotes the functionality that is relevant to the user of the service, without burdening the user with irrelevant details on how the service is implemented. A user is not interested in who provides the service. Many different providers may implement the same service, differentiating among each other through the cost/performance ratios achieved by their implementations. Therefore, services are advertised in cyberspace by service descriptions, each one capturing information on essential characteristic of a service and on the locations of associated providers. Special infrastructure functions can find and evaluate a set of service descriptions that have (near) match to a user request, thus supporting automated discovery, selection, invocation and even composition of services. The SOA paradigm is very attractive, since it promises a “lego approach” to software applications. However, the technological state-of-the art (Web Services) so far offers only a partial implementation of the SOA potential, and design approaches with clear architectural guidelines and incorporating the business view are still subject of research. This talk will discuss one such approach and consider architectural styles that play a role in it.

BRIEF BIOGRAPHY

Marten J. van Sinderen is an associate professor at the University of Twente, The Netherlands, and manager of A-Services Internet, one of the strategic research orientations of the Centre for Telematics and Information Technology, the ICT research institute of the University of Twente. His research interests include design methods and architectures for networked systems, and service platforms for supporting context-aware mobile applications. He currently leads the Dutch Freeband A-MUSE project on service design and semantic interoperability. He received his Master’s degree in electrical engineering and his Ph.D. degree in computer science from the University of Twente, The Netherlands.
Programming Languages
Full Papers
Keywords: Java, concurrent object-oriented language, small-step semantics, core calculus, implementation by translation.

Abstract: We introduce state classes, a construct to program objects that can be safely concurrently accessed. State classes model the notion of object's state (intended as some abstraction over the value of fields) that plays a key role in concurrent object-oriented programming (as the state of an object changes, so does its coordination behavior). We show how state classes can be added to Java-like languages by presenting STATEJ, an extension of JAVA with state classes. The operational semantics of the state class construct is illustrated both at an abstract level, by means of a core calculus for STATEJ, and at a concrete level, by defining a translation from STATEJ into JAVA.

1 INTRODUCTION

The notion of object’s state, intended as some abstraction on the values of fields, plays a key role in concurrent object-oriented programming. Various language constructs for expressing object’s state abstractions have been proposed in the literature (see, e.g., (Philippsen, 2000) for a survey). We propose state classes, a programming feature that could be added to JAVA-like programming languages. The main novelties in our proposal are: (1) The ability of states to carry values, thanks to the fact that states may be parameterized by special fields, that we call attributes; and (2) The presence of a static type and effect system guaranteeing that, even though the state of the objects may vary through states with different attributes, no attempt will be made to access non-existing attributes (this is, for state attributes, the standard requirement that well typed programs cannot cause a field not found error).

This paper focuses on the dynamic semantics of state classes. Typing issues are addressed in (Damiani et al., 2006). The paper is organized as follows: Section 2 introduces STATEJ, an extension of JAVA with state classes, through an example. Section 3 gives the FSJ calculus (a core calculus for STATEJ). Section 4 outlines how STATEJ can be implemented by translation into plain JAVA. Sections 5 and 6 conclude by discussing related and further work, respectively.

2 AN EXAMPLE

In this section we motivate STATEJ through an example. The state class construct is designed to program objects that can be safely concurrently accessed. Therefore, in a state class, all the fields are private and all the methods are synchronized (that is, they are executed in mutual exclusion on the receiver object). A state class may extend an ordinary (i.e., non-state) class, but only state classes may extend state classes. Each state class specifies a collection of states. Each state is parameterized by some special fields, called attributes, and declares some methods. The state of an object o can be changed only inside methods of o, by means of a state transition statement, this!S(e1, . . . , en), where “S” is the name of the target state and “e1, . . . , en” (n ≥ 0) supply the values for all the attributes of S. An object belonging to a state class is always in one of the states specified in its class. Each state class constructor must set the state of the created object. The default constructor of the root of a hierarchy of state classes sets the state to the first state defined in the class.

The class ReaderWriter (in Fig. 1) implements a multiple reader, single-writer lock — see (Birrel, 1989), for an implementation using traditional concurrency primitives in a dialect of MODULA 2, and (Benton et al., 2004), for an implementation using chords in POLYPHONIC C♯.
public state class ReaderWriter {
    state FREE {
        public void shared() {this!!SHARED(1);}
        public void exclusive() {this!!EXCLUSIVE;}
    }
    state SHARED(int n) {
        public void shared() {n++;}
        public void releaseShared()
        {n--; if (n==0) this!!FREE;}
    }
    state EXCLUSIVE {
        public void releaseShared()
        {this!!FREE;}
    }
    state PENDING_WRITER(int n) {
        public void exclusive()
        {this!!PENDING_WRITER(n); this!!EXCLUSIVE;}
    }
}

Figure 1: A multiple-reader, single-writer lock.

public state class ReaderWriterFair
    extends ReaderWriter {
    state SHARED(int n) {
        public void exclusive()
        {this!!PENDING_WRITER(n); pre_exclusive();
        this!!EXCLUSIVE;}
    }
    state PENDING_WRITER(int n) {
        public void releaseShared()
        {n--; if (n==0) this!!PRE_EXCLUSIVE;}
    }
    state PRE_EXCLUSIVE {
        private void pre_exclusive() { }
    }
}

Figure 2: A fair multiple-reader, single-writer lock.

When a thread e invokes a method m on an object o belonging to a state class (e.g., to the class ReaderWriter in Fig. 1), if either o is in a state that does not support the invoked method (e.g., shared invoked on an EXCLUSIVE ReaderWriter) or some other thread is executing a method on o, then the execution of e is blocked until o reaches (because of the action of some other thread) a state where the invoked method is available and no other thread is executing a method on o.

The policy implemented by the ReaderWriter class above is prone to writers’ starvation. The class ReaderWriterFair (in Fig. 2) extends the class ReaderWriter to implement a writer starvation free policy.

An extending class inherits all the states of the extended class, and may add/override methods and introduce new states. Thus, class ReaderWriteFair has states FREE, SHARED, EXCLUSIVE, PENDING_WRITER and PRE_EXCLUSIVE. When the request exclusive is received by an object o in state SHARED(n), then the state of o is set to PENDING_WRITER(n) and the method body suspends; in this state o can only execute up to n requests of releaseShared; after the n-th such request, the state of o is set to PRE_EXCLUSIVE; in state PRE_EXCLUSIVE the method body for exclusive can continue, and will set the state of o to EXCLUSIVE.

The ReaderWriterFair class illustrates a common pattern in state class programming: the private method pre_exclusive has an empty body, and acts as a test that the receiver has reached the state PRE_EXCLUSIVE.

3 A CALCULUS FOR STATEJ

This section gives syntax and operational semantics of FSJ, a minimal imperative core calculus for STATEJ. FSJ models the innovative features of the state construct (namely state classes, state attributes and methods, and state transitions) and multi-threaded computations.

A FSJ program consists of a set of class definitions plus an expression to be evaluated, that we will call the main expression of the program.

3.1 Syntax

The abstract syntax of FSJ class declarations (L), class constructor declarations (K), state declarations (N), method declarations (M), and expressions (E) is given in Fig. 3. The metavariables A, L, C, and D range over class names; S ranges over state classes; K and f range over method names; x ranges over method parameter names; and a, b, c, d, and e range over expressions.

We write “e” as a shorthand for a possibly empty sequence “e_1,  · · · , e_n” (and similarly for C, f, S, x) and write “N” as a shorthand for “N_1 · · · N_n” with no commas (and similarly for f). We write the empty sequence as “•” and denote the concatenation of sequences using either comma or juxtaposition, as appropriate. We abbreviate operations on pair of sequences by writing “C f” for “C_1 f_1,  · · · , C_n f_n”, where n is the length of C and f. We assume that sequences of state declarations or names, attribute declarations or names, method parameter declarations or names, method declarations do not contain duplicate names.

The class declaration

\[ \text{state class } C \text{ extends } D \{ K N \} \]

defines a state class of name C with superclass D. The new class has a single constructor K and a set of states N. The state declarations N may either refine (by adding/overriding methods) states that are already present in D or add new states.
The constructor declaration \( C(\overline{f}) \{ \text{this}!S(\overline{f}) \} \) specifies how to initialize the state and the state attributes of an instance of \( C \). It takes exactly as many parameters as there are attributes of the state \( S \) and its body consists of a state transition statement.

The state declaration \( \text{state} \ S(\overline{f}) \{ \}\ ) introduces a state with name \( S \) and attributes of names \( \overline{f} \) and types \( \overline{C} \). The declaration provides a suite of methods \( \overline{R} \) that are available in the state \( S \) of the class \( C \) containing the state declaration. A state \( S \) declared in a class \( C \) inherits all the (not overridden) methods that are defined in the (possible) declarations of \( S \) contained in the superclasses of \( C \).

The method declaration \( m \ A(\overline{x}) \{ \overline{e} \} \) introduces a method named \( m \) with result type \( C \), parameters \( \overline{x} \) of types \( C \), and body \( \overline{e} \). The variables \( \overline{x} \) and the pseudo-variable this are bound in \( \overline{e} \).

The class declarations in a program must satisfy the following conditions: (1) \( \text{Object} \) is a distinguished class name whose declaration does not appear in the program; (2) For every class name \( C \) (except \( \text{Object} \)) appearing anywhere in the program, one and only one class with name \( C \) is declared in the program; and (3) The subtype relation induced by the class declarations in the program (denoted by \( < \) and formally defined in the middle of Fig. 3) is acyclic. To simplify the notation in what follows (as in (Igarashi et al., 2001)), we always assume a fixed program.

The lookup functions are given at the bottom of Fig. 3. We write \( S \not\in \overline{R} \) to mean that no declaration of the state \( S \) is included in \( \overline{R} \), and \( m \not\in \overline{R} \) to mean that no declaration of the method \( m \) is included in \( \overline{R} \). Lookup of the attributes of a state \( S \) of a class \( C \), written \( \text{attributes}(C, S) \), returns a sequence \( \overline{C} \) pairing the type of each attribute declared in the state with its name. Lookup of the definition of the method \( m \) in the state \( S \) of a state class \( C \) is denoted by \( m\text{Def}(m, C, S) \).^1

Note that \( \text{attributes}(C, S) \) and \( m\text{Def}(m, C, S) \) are undefined when \( C = \text{Object} \).^2

### 3.2 Operational Semantics

In this section we introduce the operational semantics of FSJ, by defining the reduction rules that transform configurations representing multi-threaded computation. A configuration is a pair \( \langle \bar{e}, \mathcal{H} \rangle \), where \( \bar{e} \) is a sequence of \( n \geq 1 \) runtime expressions and \( \mathcal{H} \) is a heap mapping addresses to objects. Addresses, ranged over by the metavariable \( \bar{t} \), are the elements of the denumerable set \( \bar{I} \). Objects are finite mappings associating: (1) the distinguished name “class” to a class name indicating the class of the object; (2) the distinguished name “state” to a state name indicating the state of the object; and (3) a mapping associating a finite number (possibly zero) of state attribute names to addresses. Objects will be denoted by \( \bar{J} : \bar{C}, \bar{S}, \bar{f}, \bar{t} \).

The first component of a configuration, \( \bar{e} \), will be called “sequence of threads”. A thread of computation is represented by the evaluation of a runtime expression \( e_i \) in the heap \( \mathcal{H} \). The different threads share the same heap \( \mathcal{H} \). Threads do not have, as in full STATEJ and JAVA, an associated stack, keeping the association between parameters and values. In fact, since FSJ does not include assignment, method calls are evaluated by directly substituting the formal parameters and the metavariable \( \text{this} \) with the corresponding values (in FSJ the only values are addresses). We call the result of this substitution, which is no longer an expression of the source language, a simple runtime expression. Simple runtime expressions, ranged over by \( s \), are obtained from the pseudo grammar defining expressions (in Fig. 3) by replacing the clauses “\( \bar{x} \mid \text{this} \mid \text{this.f} \)” with the clauses “\( \bar{t} \mid \bar{f} \mid \bar{t}.\bar{f} \)” (see the top of Fig. 4).

Runtime expressions, ranged over by \( e \), are defined by the grammar at top of Fig. 4. In FSJ every method is synchronized, therefore on method call the lock of the object receiving the call must be ac-

---

^1In full STATEJ, like in JAVA, the lookup functions take into account method overloading, that (for simplicity) is not included in FSJ.

^2In full STATEJ the class Object has several methods.
Simple runtime expressions, runtime expressions, evaluation contexts, redexes, and auxiliary functions:

\[
\begin{align*}
\text{s} & ::= e \mid e.f \mid s.s \mid \text{new C}(s) \mid \text{unlock}(s) \mid \text{spawn}(s) \mid s.m(s) \\
\text{e} & ::= e \mid e.f \mid e:c \mid s.s \mid \text{new C}(e, c) \mid \text{unlock}(e) \mid \text{spawn}(e) \mid e.m(s) \mid s.m(e, c, e) \\
\text{E} & ::= [] \mid \text{E}.s \mid \text{new C}(e, E, s) \mid \text{unlock}(E) \mid \text{spawn}(E) \mid E.m(s) \mid e.m(e, c, s) \mid \text{ret}(e, n, E) \\
\text{r} & ::= e.f \mid e:s \mid \text{new C}(e) \mid \text{unlock}(e) \mid \text{spawn}(e) \mid e.m(i) \mid \text{ret}(e, m, r) \mid \text{unlock}(e.m(i))
\end{align*}
\]

\[
\begin{align*}
\text{lockedBy}(e) & = \{ i \mid \text{ret}(i, \cdots, \cdots) \text{ is a subexpression of } e \text{ and } \text{unlock}(i, \cdots, \cdots) \text{ is not a subexpression of } e \} \\
\text{lockedBy}(e_1 \cdots e_n) & = \bigcup_{1 \leq i \leq n} \text{lockedBy}(e_i)
\end{align*}
\]

Reduction rules:

\[
\begin{align*}
\text{R-Attr} & : \frac{\text{H}(i) = o \ \text{and} \ o(t) = i'}{\text{a E}[i.f, E \rightarrow a E[i', E, \text{H}]}}, \\
\text{R-Seq} & : \frac{\text{a E}[i; s, E \rightarrow a E[s, E, \text{H}]} \text{ and } (i \notin \text{Dom}(\text{H}))}{\text{R-New}} \\
\text{R-Trans} & : \frac{\text{H}(i)(\text{class}) = C \ \text{attributes}(C, S) = \text{C} f \ o = \text{[class : C, state : S, f : i]} \text{ and } (i \notin \text{Dom}(\text{H}))}{\text{a E}[\text{new C}(i), E \rightarrow a E[i, E, \text{H}[i : o]]}, \\
\text{R-Spawn} & : \frac{\text{a E}[\text{spawn}(i), E \rightarrow a E[i, E, i.runt], \text{H}]}{\text{R-Invk-1}} \\
\text{R-Invk-2} & : \frac{\text{i \notin \text{lockedBy}(a \circ t)} \text{ and } \text{H}(i) = \text{[class : D, state : S, \cdots]} \ \text{mDef}(D, D, S) = C.m(C \ x) \ {\{e}\}}{\text{a E}[\text{m[i]}, E \rightarrow a E[i, E, \text{H}[i : o]]}}, \\
\text{R-Unlock} & : \frac{\text{i \notin \text{lockedBy}(a \circ t)} \text{ and } \text{H}(i) = \text{[class : D, state : S, \cdots]} \ \text{mDef}(D, D, S) = C.m(C \ x) \ {\{e}\}}{\text{a E}[\text{unlock}(i, m[i]), E \rightarrow a E[i, E, \text{H}[i : o]]}}, \\
\text{R-Ret} & : \frac{\text{a E}[\text{ret}(i, m[i]), E \rightarrow a E[i, E, \text{H}[i : o]]]}{{}}
\end{align*}
\]

Figure 4: FSJ (simple) runtime expressions, evaluation contexts, redexes, auxiliary functions, and reduction rules.

required, unless the call is inside a method of the object itself, in which case the call can proceed (the lock is reentrant). Moreover, when the method call is on a method not defined in the current state, the lock of the object must be released. This gives to other threads a chance to change the state of the object to a state in which the method is defined. Both these situations are modelled by particular runtime expressions:

1. \text{ret}(i, m, c), where \( e \) does not contain occurrences of unlock(\( i, \cdots, \cdots \)), specifies that a thread is currently holding the lock of the receiver \( i \), in order to evaluate the expression \( e \), which represents the body of the method \( m \), and \( \text{unlock}(i, m[i]) \) specifies that the lock of \( i \) has been released in order to give a chance to another thread to change the state of \( i \) to a state in which \( m \) is defined. Note that, the definition of the syntax for runtime expressions implies that there can be nested \text{ret} expressions but only one \text{unlock}.

The metavariables \( a, b, c, d \), and \( e \) range over runtime expressions. We write \( a \) as a shorthand for a possibly empty sequence \( a_1 \cdots a_n \) and \( a \) as a shorthand for a possibly empty sequence of length almost one. The function \( \text{lockedBy}(e) \), defined in Fig. 4, returns the set of addresses that are locked by the thread sequence \( e \).

The reduction relation has the form \( \alpha b_1 \ e. \bar{H}_1 \rightarrow \alpha b_2 \ e. \bar{H}_2 \), read “configuration \( \alpha b_1 \ e. \bar{H}_1 \) reduces to configuration \( \alpha b_2 \ e. \bar{H}_2 \) in one step”. The (empty or singleton) sequence \( \bar{d} \) indicates that a new thread might have been spawned because of the reduction of a \text{spawn} expression. We write \( \rightarrow^* \) for the reflexive and transitive closure of \( \rightarrow \).

By using the definition of evaluation context and redex (see \( E \) and \( r \) in Fig. 4), the reduction rules ensure that inside each thread the computation follows a call-by-value left-to-right reduction strategy. This implies that expressions such as \text{ret} and \text{unlock} can only be preceded by values and followed by simple runtime expressions, which do not contain \text{ret} and \text{unlock} (see the definition of \( s \) and \( e \) in Fig. 4).

The following property asserts that a context can be decomposed in a unique way in sub-contexts showing the activation stack of method calls.

**Property 1 (Unique Decomposition)** Every evaluation context \( \bar{c} \) in the reduction relation can be decomposed into a sequence of runtime expressions, which do not contain \text{ret} and \text{unlock}.
tion context $E$ can be written as

$$
E_{1,1}[\text{ret}(i, m_1, 1, \ldots, E_{1,q_1}[\text{ret}(i, m_{1,q_1}, \ldots, E_p, q_p, [\text{ret}(i_p, m_{p,q_p}, E_0)] \ldots, \ldots, \ldots)]), \ldots, \ldots, \ldots]
$$

where $E_{1,1}, \ldots, E_{1,q_1}, \ldots, E_p, q_p, (p \geq 0, q_1 \geq 1, \ldots, q_p \geq 1)$ and $E_0$ do not contain $\text{ret}(\ldots)$ subexpressions.

The reduction rules are given at the bottom of Fig. 4. Each reduction rule rewrites a configuration of the form "$aE[r] \cdot d, H_1$", where $E$ is an evaluation context and $r$ is a redex, into a configuration of the form "$aE[e] \cdot d, H_2$". The metavariable $o$ ranges over objects. We use $H[t : o]$ to denote the heap such that $H[0] = o$ and $H[t : o](t') = H(t')$, for $t' \neq t$.

The reduction rules for attribute selection, (R-ATTR), and sequential composition, (R-SEQ), are standard. The rule for object creation, (R-NEW), stores the newly created object at a fresh address of the heap and returns the address. The pseudo fields class and state, and the parameters of the initial state are initialized as specified by the class constructor. The rule for state transition, (R-TRANS), changes the current state of the object and returns its address. Rule (R-SPAWN) replaces the spawn expression with the address $t$ and adds a new thread evaluating the call of the method run on the object at $t$. Rule (R-INVK) is applied if the method $m$ is defined in the current state of the receiver, $t$, and no other thread holds the lock of $t$. The expression produced replaces the call with $\text{ret}(t, \cdot, m, t')$, indicating that the current thread holds the lock of $t$. The expression $e'$ is the body of the method $m$ in which this and the formal parameters are replaced with the address $t$ and the actual parameters. Rule (R-INVK-2) is applied if the method $m$ is not defined in the current state of the receiver and the current thread holds the lock of $t$. In this case, the lock of $t$ must be released and the thread must wait that some other thread changes the state of $t$ to a state in which $m$ is defined. This is achieved by replacing the method call redex with the expression $\text{unlock}(t, \cdot, m(t))$. Note that, since the current thread had the lock of $t$, the newly introduced unlock expression is a subexpression of an expression $\text{ret}(t, m', t')$ for some $m'$ and $t'$. Rule (R-UNLOCK) replaces the expression $\text{unlock}(t, \cdot, m(t))$, if $t$ is not locked and the method $m$ is defined in its state, with $\text{ret}(t, m, t')$, where $t'$ is the body of the method $m$ in which this and the formal parameters are replaced with the address $t$ and the actual parameters. Rule (R-RET), that applies when the body of the method $m$ on object $t$ has been evaluated completely, producing a value, releases the lock of $t$ by removing the $\text{ret}(t, m, t)$ subexpression.

Example 2 (Application of the reduction rules)

First we define the following classes $CR$ and $CW$ representing the class of threads that have a shared access to a ReaderWriter object $rw$ and the class of threads that have an exclusive access to it, respectively.

state class $CR$ extends Object {
    CR(ReaderWriter rw) { this!!S(rw) }
    state S (ReaderWriter rw) {
        Object run () {
            rw.shared();
            ...
            rw.releaseShared();
            this.run() }
    }
}

state class $CW$ extends Object {
    CW(ReaderWriter rw) { this!!S(rw) }
    state S (ReaderWriter rw) {
        Object run () {
            rw.exclusive();
            ...
            rw.releaseExclusive();
            this.run() }
    }
}

We consider as the main expression of the program, that is the expression to be evaluated,

$$\text{spawn}(\text{new} CR(t)); (\text{new} CR(t)).run(),$$

where $t$ is a ReaderWriter object, so the computation starts from the following configuration:

$$\text{spawn}(\text{new} CR(t)); (\text{new} CR(t)).run(), H$$

where $H = \{ [\text{class}: \text{ReaderWriterFair}, \text{state}: \text{FREE}] \}$.

A possible computation is as in Fig. 5, where $SH$ stands for SHARED, $P4$ stands for PENDING_WRITER, $EX$ stands for EXCLUSIVE, and $PE$ stands for PRE_EXCLUSIVE. We adopt the following notations: (1) Threads $e_1, e_2$ being part of the configuration are written $(e_1); (2)$ Redexes are underlined; (3) Redexes of suspended threads are underlined and written in grey; (4) the arrow $\Rightarrow$ indicates one step of reduction for each thread of the sequence; (5) In $\text{ret}$ expressions we omit method names. As we see in Fig. 5, in the example we assumed to have integers, decrement and if-statement. These are assumed, in line $(\#)$, to be reduced following the standard semantics.

4 FROM $\text{STATEJ}$ TO $\text{JAVA}$

This section briefly illustrates a translation from $\text{STATEJ}$ to plain $\text{JAVA}$. The basic idea of the translation is to map a state class into a $\text{JAVA}$ class using synchronized methods and the primitives $\text{wait}()$ and $\text{notify}()$. A class contains a field indicating the
current state of the object, and methods corresponding to the methods of the original STATEJ class. The translation can be briefly described as follows.

**Method.** Methods defined in more than one state have more than one body. To be able to execute different bodies in different states our translation creates a unique synchronized method containing all the different bodies. At run-time, when the method is called, we have to check the current state of the object, and see whether the method was defined in this state or not. In case it is defined, then the corresponding body is executed, otherwise the thread calls a wait() putting it in hold. To keep the information on the methods defined in a certain state we use a hash table. Due to the limitation of the switch statement of JAVA, states are codified by the primitive type int. For example the following class

```java
state class Ex extends Object {
    Ex() { this!=A(); }
    state A () {
        Object m() { /* body of m in A */ }
        state B () { /* body of m in B */ }
    }
}
```

is translated into

```java
class Ex extends Object {
    Ex() { ... }  
    final static int A = 1;  
    final static int B = 2;  
    int currentState;  
    synchronized Object m() {  
        while (!existsInCurrentState) wait();  
        switch (currentState) {  
            case A: /* body of m in A */ break;  
            case B: /* body of m in B */ break;  
        }  
    }
}
```

where the existence of a method in a given state and its selection are done using the hash table of methods.

**State Transition.** The state transition expression this!=A() is translated into
so in addition to change the state of the objects it notifies all the threads waiting for the lock of the current object. When the current thread will release the lock the notified threads will compete to get it to have a chance to see whether the method that caused the waiting is defined in the current state. If the method is defined, then the thread can proceed, otherwise it calls a `wait()`. Due to the non deterministic nature of JAVA scheduling we cannot insure the order in which notified threads will be woken up.

**Constructor.** The constructor of the translated class should initialize the hash table and then include the translation of the constructor of the original class.

**Inheritance.** A state class may extend another class (either state or not). In the subclass we inherit all the states and may add others. Therefore, we have to be careful to clashes of constants of state. Moreover, methods may be added/redefined. For instance method `exclusive()` of the example in Sect. 2, is defined in state `FREE` of `ReaderWriter`, and redefined in state `SHARED` of `ReaderWriterPair`. When a method is redefined in its translation we use the `default` clause as follows.

```java
class ReaderWriterPair
    extends ReaderWriter {
      ...
      synchronized void exclusive () {
        while (!existsInCurrentState) wait();
        switch (currentState) {
          case SHARED:
            currentState = PENDING_WRITER;
            notifyAll();
            pre_exclusive();
            currentState = EXCLUSIVE;
            notifyAll();
            break;
          default :
            super.exclusive;
            break; }
      }
    }
```

The current implementation of the translator (www.di.unito.it/~giannini/stateJimpl/) takes as input a program written in JAVA 1.4 extended with state classes *with attribute-free states* (attributes can be straightforwardly codified by class fields; however, their implementation would require to implement the type and effect analysis). The translation uses the tool for Language Recognition ANTLR, see (Parr and project group, 2005), and the StringTemplate technology, see (Parr, 2005). We first made a JAVA 1.4 to JAVA 1.4 translation taking advantage of the grammar defined by Parr and then modified the grammar to include our state related constructs. The use of ANTLR and StringTemplate makes the translator easily adaptable to different translation schemes and also to addition to the input language.

### 5 RELATED WORK

According to (Philippsen, 2000) *states provide a boundary coordination* mechanism (we refer to Sect. 4.2 of (Philippsen, 2000) for a survey of several COOLs with boundary coordination). In particular, the state class construct is related to the *actor model* (Agha, 1986) and to the *behaviour abstraction* and *behaviour/enable sets* proposals (Kafura and Lavender, 1996; Tomlinson and Singh, 1989).

At the best of our knowledge, the main novelties in our proposal are: the ability of states to carry values (thanks to the presence of attributes); the formalization of an abstract operational semantics of a notion of state for expressing coordination in JAVA-like languages; and the presence of a static type and effect system (presented in (Damiani et al., 2006)) guaranteeing that during the execution there cannot be any access to undefined attributes of objects. Type systems for concurrent objects have been investigated in the literature, see, e.g., “regular object types” (Nierstrasz, 1993), the TyCO object calculus (Ravara and Vasconcelos, 2000), and the *FickletAT* proposal (Damiani et al., 2004).

Various improvements of the concurrency model of JAVA-like languages have been proposed. In **JOIN JAVA** (Itzstein and Kearney, 2001) and **POLYPHONIC C♯** (Benton et al., 2004) the synchronization mechanism relies on the *join pattern*, called *chord* in **POLYPHONIC C♯**, construct. Chords can be used to codify the state of an object through the pattern (illustrated, for instance, in (Benton et al., 2004)) of using private asynchronous method to carry object state. However, this pattern could be misused leading to deadlock or errors. In **STATEJ** the notion of object state is in the language definition, thus eliminating the possibility of many of such errors. In **JEEN** (Milicia and Sassone, 2005) the synchronization conditions on an object o are expressed with linear temporal logic constraints involving the value of fields and the method invocation history of o. These constraints could be used to codify the state of an object o. However, state attributes have to be mapped on object fields and there is no way to express the fact that some fields should be accessible only in some states.

**STATEJ** (as **JOIN JAVA**, **POLYPHONIC C♯**, and **JEEN**) focuses on a specific coordination mechanism. The JR programming language (Keen et al., 2004) takes a different approach: it extends JAVA providing a rich concurrency model with a variety of mechanisms. None of these mechanisms directly models the notion of object state.
6 FUTURE WORK

The current prototypical implementation of STATEJ (www.di.unito.it/~giannini/stateJimpl/) is based on the translation scheme outlined in Sect. 4. It consists of a preprocessor that maps code written in JAVA 1.4 extended with state classes into plain JAVA. The current approach favors simplicity over efficiency. Its major drawback is that each state transition of an object o notifies all the threads waiting for any state of o. Note that, notifying just the threads waiting for the target state of the transition would not represent a significative improvement, since multiple state transitions may occur before the lock on o is released. A more significative improvement would be moving notification from state transition on o to lock release on o: this would allow notifying just the threads waiting for the current state of o. Note that, however, all the first (according to the scheduling mechanism of JAVA) of such threads have to sleep again. We are currently investigating a quite different approach that support selective wakeups. It can be roughly described as follows:

- Each object o is equipped with a set of FIFO queues (one for each state).
- Whenever a thread invokes a method m on o, IF o is locked by some other thread OR m is not available in the current state of o
  - THEN the thread is suspended and enqueued in all the queues associated to the states of o where m is available, and the lock on o (if held by the suspended thread) is released
  - ELSE the method executed and the lock on o (if not already held by the invoking thread) is taken.
- Whenever the lock on o is released, IF the queue associated to the current state of o is not empty, THEN a thread o is extracted from the queue, removed from all the other queues, resumed, and it takes the lock on o.

Other future work includes: Refinement of the type and effect system given in (Damiani et al., 2006); Further investigations on the expressivity of the state class construct and on its integration in JAVA-like languages (by analyzing the interaction of state classes and their types with the advanced features of JAVA-like languages); Development of a new prototype (based on the translation scheme outlined above) including state attributes and the related type and effect analysis; and Development of benchmarks.


Igarashi, A., Pierce, B., and Wadler, P. (2001). Featherweight Java: A minimal core calculus for Java and GJ. ACM TOPLAS, 23(3):396–450.


REFERENCES

SOFTWARE IMPLEMENTATION OF THE IEEE 754R DECIMAL FLOATING-POINT ARITHMETIC

Marius Cornea, Cristina Anderson, Charles Tsen
Intel Corporation

Keywords: IEEE 754R, IEEE 754, Floating-Point, Binary Floating-Point, Decimal Floating-Point, Basic Operations, Algorithms, Financial Computation, Financial Calculation.

Abstract: The IEEE Standard 754-1985 for Binary Floating-Point Arithmetic (IEEE Std. 754, 1985) is being revised (IEEE Std. 754R Draft, 2006), and an important addition to the current text is the definition of decimal floating-point arithmetic (Cowlishaw, 2003). This is aimed mainly to provide a robust, reliable framework for financial applications that are often subject to legal requirements concerning rounding and precision of the results in the areas of banking, telephone billing, tax calculation, currency conversion, insurance, or accounting in general. Using binary floating-point calculations to approximate decimal calculations has led in the past to the existence of numerous proprietary software packages, each with its own characteristics and capabilities. New algorithms are presented in this paper which were used for a generic implementation in software of the IEEE 754R decimal floating-point arithmetic, but may also be suitable for a hardware implementation. In the absence of hardware to perform IEEE 754R decimal floating-point operations, this new software package that will be fully compliant with the standard proposal should be an attractive option for various financial computations. The library presented in this paper uses the binary encoding method from (IEEE Std. 754R Draft, 2006) for decimal floating-point values. Preliminary performance results show one to two orders of magnitude improvement over a software package currently incorporated in GCC, which operates on values encoded using the decimal method from (IEEE Std. 754R Draft, 2006).

1 INTRODUCTION

Binary floating-point arithmetic can be used in most cases to approximate decimal calculations. However errors may occur when converting numerical values between their binary and decimal representations, and errors can accumulate differently in the course of a computation depending on whether it is carried out using binary or decimal floating-point arithmetic. For example, the following simple C program will not have in general the expected output b=7.0 for a=0.0007.

```c
main () {
  float a, b;
  a = 7/10000.0;
  b = 10000.0 * a;
  printf ("a = %x = %10.10f
", *(unsigned int *)&a, a);
  printf ("b = %x = %10.10f
", *(unsigned int *)&b, b);
}
```

(The value 7.0 has the binary encoding 0x40e00000.) The actual output on a system that complies with the IEEE Standard 754 will be:

- a = 3a378034 = 0.0007000000
- b = 40dfefef = 6.999997504

Such errors are not acceptable in many cases of financial computations, mainly because legal requirements mandate how to determine the rounding errors - in general following rules that humans would use when performing the same computations on paper, and in decimal. Several software packages exist and have been used for this purpose so far, but each one has its own characteristics and capabilities such as precision, rounding modes, operations, or internal storage formats for numerical data. These software packages are not compatible with each other in general. The IEEE 754R standard proposal attempts to resolve these issues by defining all the rules for decimal floating-point arithmetic in a way that can be adopted and implemented on all computing systems in software, in hardware, or in a combination of the...
two. Using IEEE 754R decimal floating-point arithmetic, the previous example could then become:

```c
main () {
    decimal32 a, b;
    a = 7/10000.0;
    b = 10000.0 * a;
    printf ("a = %x = %10.10fd\n", *(unsigned int *)&a, a);
    printf ("b = %x = %10.10fd\n", *(unsigned int *)&b, b);
}
```

(The hypothetical format descriptor %fd is used for printing decimal floating-point values.) The output on a system complying with the IEEE Standard 754R proposal would then represent the result without any error:

```
a = 30800007 = 0.0007000000
b = 32800007 = 7.0000000000
```

(The IEEE 754R binary encoding for decimal floating-point values was used in this example.) The following section summarizes the most important aspects of the IEEE 754R decimal floating-point arithmetic definition.

## 2 IEEE 754R DECIMAL FLOATING-POINT

The IEEE 754R standard proposal defines three decimal floating-point formats with sizes of 32, 64, and 128 bits. Two encodings for each of these formats are specified: a decimal-based encoding which is best suited for certain possible hardware implementations of the decimal arithmetic (Erle et al., 2005), and a binary-based encoding better suited for software implementations on systems that support the IEEE 754 binary floating-point arithmetic in hardware (Tang, 2005). The two encoding methods are otherwise equivalent, and a simple conversion operation is necessary to switch between the two.

As defined in the IEEE 754R proposal, a decimal floating-point number $n$ is represented as

$$n = \pm C \times 10^e$$

where $C$ is a positive integer coefficient with at most $p$ decimal digits, and $e$ is an integer exponent. A precision of $p$ decimal digits will be assumed further for the operands and results of decimal floating-point operations.

Compared to the binary single, double, and quad precision floating-point formats, the decimal floating-point formats denoted here by decimal32, decimal64, and decimal128 cover different ranges and have different precisions, although they have similar storage sizes. For decimal, only the wider formats are used in actual computations, while decimal32 is defined as a storage format only. For numerical values that can be represented in these binary and decimal formats, the main parameters that determine their range and precision are shown in Table 1.

<table>
<thead>
<tr>
<th>Table 1: IEEE 754 binary and IEEE 754R decimal floating-point format parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Binary Formats</strong></td>
</tr>
<tr>
<td><strong>Prec.</strong></td>
</tr>
<tr>
<td>$E_{\text{min}}$</td>
</tr>
<tr>
<td>$E_{\text{max}}$</td>
</tr>
<tr>
<td><strong>Decimal Formats</strong></td>
</tr>
<tr>
<td><strong>Prec.</strong></td>
</tr>
<tr>
<td>$E_{\text{min}}$</td>
</tr>
<tr>
<td>$E_{\text{max}}$</td>
</tr>
</tbody>
</table>

The following sections will present new algorithms that can be used for an efficient implementation in software of the decimal floating-point arithmetic as defined by the IEEE 754R proposal. Mathematical proofs of correctness have been developed, but will not be included here for brevity. Compiler and runtime support libraries could use the implementation described here, which addresses the need to have a good software solution for the decimal floating-point arithmetic.

## 3 CONVERSIONS BETWEEN DECIMAL AND BINARY FORMATS

In implementing the decimal floating-point arithmetic defined in IEEE 754R, conversions between decimal and binary formats are necessary in many situations.

For example, if decimal floating-point values are encoded in a decimal-based format (string, BCD, IEEE 754R decimal encoding, or other) they need to be converted to binary before a software implementation of the decimal floating-point operation can take full advantage of the existing hardware for binary operations. This conversion is relatively easy to implement, and should exploit any available instruction-level parallelism.
The opposite conversion, from binary to decimal format may have to be performed on results before writing them to memory, or for printing in string format decimal numbers encoded in binary. Another reason for binary-to-decimal conversion could be for rounding a decimal floating-point result to a pre-determined number of decimal digits, if the exact result was calculated first in binary format. The straightforward method for this is to convert the exact result to decimal, round to the destination precision and then, if necessary, convert the coefficient of the final result back to binary. This step can be avoided completely if the coefficients are stored in binary.

The mathematical property presented next was used for this purpose. It gives a precise way to ‘cut off’ x decimal digits from the lower part of an integer C when its binary representation is available, thus avoiding the need to convert C to decimal, remove the lower x decimal digits, and then convert the result back to binary. This property was applied to conversions from binary to decimal format as well as in the implementation of the most common decimal floating-point operations: addition, subtraction, multiplication, fused multiply-add, and in part, division.

For example if the decimal number C = 123456789 is available in binary and its six most significant decimal digits are required, Property 1 specifies precisely how to calculate the constant $k_x$ = $10^{33.3}$ so that $\lceil C \cdot k_x \rceil$ = 123456, with certainty, while using only the binary representation of C. The values $k_x$ are pre-calculated. (Note: the floor(x), ceiling(x), and fraction(x) functions are denoted here by $\lfloor x \rfloor$, $\lceil x \rceil$, and $\{ x \}$ respectively.)

**Property 1.**

Let C be a number in base $b = 2$ and $C = d_0 \cdot 10^{e_0} + d_1 \cdot 10^{e_1} + \ldots + d_q \cdot 10^{-q}$ its representation in base B=10, where $d_0$, $d_1$, ..., $d_q$ are pre-calculated. (Note: the floor(x), ceiling(x), and fraction(x) functions are denoted here by $\lfloor x \rfloor$, $\lceil x \rceil$, and $\{ x \}$ respectively.)

Let $x \in \{1, 2, 3, \ldots, q\}$ and $\epsilon = \log_{10} 10$. If $y \in \mathbb{N}$, $y \lfloor \ldots, x \rfloor + \ldots \{ q \}$ and $k_x$ is the value of $10^x$ rounded up to y bits (the subscript RP,y indicates rounding up y bits in the significand), i.e.: $k_x = (10^x)_{RP,y}$, then $\lceil C \cdot k_x \rceil$ will be precisely the desired result. The property states that $y \lceil \ldots, x \rfloor = 10^{\epsilon} \{ q \}$. However, in practice it is sufficient to take $y = \lceil 1 + \epsilon \rceil = 1 + \lceil \epsilon \rceil$ where $\lceil \epsilon \rceil$ is the ‘ceiling’ of $\epsilon$. (e.g. $\lceil 33.3 \rceil = 34$). Note that $\rho = \log_{10} 3.3219 \ldots$ and $2^{\rho} = 10$. For example if we want to remove the x lower decimal digits of a 16-digit decimal number, we can multiply the number with an approximation of $10^x$ rounded up to $y = 1 + \lceil \epsilon \rceil$, 16 = 55 bits, followed by removal of the fractional part in the product.

The relative error $\epsilon$ associated with the approximation of $10^x$ which was rounded up to y bits satisfies $0 < \epsilon < 2 \cdot 10^{-y - 1}$. The values $k_x$ for all $x$ of interest are pre-calculated and are stored as pairs $(K_x, e)$, with $K_x$ and $e$ positive integers: $k_x = K_x \cdot 2^{-e}$. This allows for implementations exclusively in the integer domain of some decimal floating-point operations, in particular addition, subtraction, multiplication, fused multiply-add, and certain conversions.

## 4 DECIMAL FLOATING-POINT ADDITION

It will be assumed that

- $n_1 = C_1 \cdot 10^{e_1}$ $C_1 \in \mathbb{Z}, 0 < C_1 < 10^9$
- $n_2 = C_2 \cdot 10^{e_2}$ $C_2 \in \mathbb{Z}, 0 < C_2 < 10^9$

are two non-zero decimal floating-point numbers with coefficients having at most p decimal digits stored as binary integers and that their sum has to be calculated, rounded to p decimal digits using the current IEEE rounding mode (this is indicated by the subscript rnd,p).

$$n = (n_1 + n_2)_{\text{rnd,p}} = C \cdot 10^{e}$$

The coefficient C needs to be correctly rounded, and is stored as a binary integer as well. For simplicity, it will be assumed that $n_1 \geq 0$ and $e_1 \geq e_2$. (The rules for other combinations of signs or exponent ordering can be derived from here.)

If the exponent $e_1$ of $n_1$ and the exponent $e_2$ of $n_2$ differ by a large quantity, the operation is simplified and rounding is trivial because $n_2$ represents just a rounding error compared to $n_1$. Otherwise if $e_1$ and $e_2$ are relatively close the coefficients $C_1$ and $C_2$ will ‘overlap’, the coefficient of the exact sum may have more than p decimal digits, and so rounding may be necessary. All the possible cases will be quantified next.

If the exact sum is $n'$, let $C'$ be the exact (not yet rounded) sum of the coefficients:

$$n' = n_1 + n_2 = C_1 \cdot 10^{e_1} + C_2 \cdot 10^{e_2} = (C_1 \cdot 10^{e_1-e_2} + C_2) \cdot 10^{e_2}$$

$$C' = C_1 \cdot 10^{e_1-e_2} + C_2$$
Let q1, q2, and q be the numbers of decimal digits needed to represent C1, C2, and C*. If not zero, the rounded coefficient C will require between 1 and p decimal digits. Rounding is not necessary if C* represented in decimal requires at most p digits, but it is necessary otherwise.

If q ≤ p, then the result is exact:
\[ n = (n')_{\text{round}} = (C' \cdot 10^{q})_{\text{round}} = (C')_{\text{round}} \cdot 10^{q} = C' \cdot 10^{q} \]

Otherwise, if q > p let x = q - p ≥ 1. Then:
\[ n = (n')_{\text{round}} = (C' \cdot 10^{q})_{\text{round}} = (C')_{\text{round}} \cdot 10^{q} = C' \cdot 10^{q+x} \]

If after rounding C = 10^n (rounding overflow), then n = 10^{p+1} \cdot 10^{q+x}\n
A simple analysis shows that rounding is trivial if q1 + e1 - q2 - e2 ≥ p. If this is not the case, i.e. if
\[ |q1 + e1 - q2 - e2| ≤ p - 1 \]
then the sum C' has to be calculated and it has to be rounded to p decimal digits. This case can be optimized by separating it in sub-cases as shall be seen further.

The algorithm presented next uses Property 1 in order to round correctly (to the destination precision) the result of a decimal floating-point addition in rounding to nearest mode, and also determines correctly the exactness of the result by using a simple comparison operation. First, an approximation of the result’s coefficient is calculated using Property 1. This will be either the correctly rounded coefficient, or it will be off by one ulp (unit-in-the-last-place). The correct result as well as its exactness can be determined directly from the calculation, without having to compute a remainder through a binary multiplication followed by a subtraction for this purpose. This makes the rounding operation for decimal floating-point addition particularly efficient.

**Decimal Floating-Point Addition with Rounding to Nearest**

The straightforward method to calculate the result is to convert both coefficients to a decimal encoding, perform a decimal addition, round the exact decimal result to nearest to the destination precision, and then convert the coefficient of the final result back to binary. It would also be possible to store the coefficients in decimal all the time, but then neither software nor hardware implementations could take advantage easily of existing instructions or circuitry that operate on binary numbers. The algorithm used for decimal floating-point addition in rounding to nearest mode is Algorithm 1, shown further.

If the smaller operand represents more than a rounding error in the larger operand, the sum C = C1 \cdot 10^{e1-e2} + C2 is calculated. If the number of decimal digits q needed to represent this number does not exceed the precision p of the destination format, then no rounding is necessary and the result is exact. If q > p, then x = q - p decimal digits have to be removed from the lower part of C', and C' has to rounded correctly to p decimal digits. For correct rounding to nearest, 0.5 ulp is added to C*: C'' = C' + 1/2 \cdot 10^p. The result is multiplied by k_x = 10^x (C*) = C' \cdot k_x, where the pre-calculated values k_x are stored for all x \{1, 2, \ldots, p\}. A test for midpoints follows (0 < f_p < 10^p, where f_p is the fractional part of C*): and if affirmative, the result is rounded to the nearest even integer. (For example if the exact result 4567.5 has to be rounded to nearest to four decimal places, the rounded result will be 4568.) Next the algorithm checks for rounding overflow (p+1 decimal digits are obtained instead of p) and finally it checks for exactness.

Note that the straightforward method for the determination of midpoints and exactness is to calculate a remainder r = C" - C \cdot 10^p \in \{0, 10^p\}$. Midpoint results could be identified by comparing the remainder with 1/2\cdot10^p, and exact results by comparing the remainder with 0. However, the calculation of a remainder – a relatively costly operation – was avoided in Algorithm 1 and instead a single comparison to a pre-calculated constant was used. This simplified method to determine midpoints and exactness along with the ability to use Property 1 make Algorithm 1 more efficient for decimal floating-point addition than previously known methods.

**Algorithm 1. Calculate the sum of two decimal floating-point numbers rounded to nearest to p decimal digits, and determine its exactness.**

q1, q2 = number of decimal digits needed to represent C1, C2 // from table lookup
if |q1 + e1 - q2 - e2| ≥ p then
  // assuming that e1 ≥ e2 round the result
  // directly as 0 < C2 < 1 ulp (C1 \cdot 10^{e1-e2});
  the result n = C1 \cdot 10^{e1} or
  n = C1 \cdot 10^{e1} ± 10^{e1+1-p} is inexact
else // if |q1 + e1 - q2 - e2| ≤ p - 1
C' = C1 \cdot 10^{e1-e2} + C2 // binary integer
  // multiplication and addition;
  // 10^{p+1} from table lookup
q = number of decimal digits needed to represent C' // from table lookup
if q ≤ p the result n = C' \cdot 10^{q} is exact
else if q ∈ [p+1, 2p] continue
x = q - p, number of decimal digits to be
removed from lower part of C', x \in \{1, p\}
C'' = C' + 1/2 \cdot 10^p \cdot 1/2 \cdot 10^p
  // pre-calculated, from table lookup
k_x = 10^x (1 + e), 0 < e < 2 \cdot 10^{p-p} \]
  // pre-calculated as specified in Property 1
C* = C'' \cdot k_x = C' \cdot k_x \cdot 2^{e2}
  // binary integer multiplication with
Decimal Floating-Point Addition when Rounding to Zero, Down, or Up

The method to calculate the result when rounding to zero or down is similar to that for rounding to nearest. The main difference is that the step for calculating \( C' = C + 1/2 \cdot 10^p \) is not necessary anymore, because midpoints between consecutive floating-point numbers do not have a special role here. For rounding up, the calculation of the result and the determination of its exactness are identical to those for rounding down. However, when the result is inexact then one ulp has to be added to it.

## 5 DECIMAL FLOATING-POINT MULTIPLICATION

It will be assumed that the product
\[
n = (n_1 \cdot n_2)_{\text{mod},p} = C \cdot 10^p
\]
has to be calculated, where the coefficient \( C \) of \( n \) is correctly rounded to \( p \) decimal digits using the current IEEE rounding mode, and is stored as a binary integer. The operands \( n_1 = C_1 \cdot 10^{e_1} \) and \( n_2 = C_2 \cdot 10^{e_2} \) are assumed to be strictly positive (for negative numbers the rules can be derived directly from here). Their coefficients require at most \( p \) decimal digits to represent and are stored as binary integers, possibly converted from a different format/encoding.

Let \( q \) be the number of decimal digits required to represent the full integer product \( C' = C_1 \cdot C_2 \) of the coefficients of \( n_1 \) and \( n_2 \). Actual rounding to \( p \) decimal digits will be necessary only if \( q \in [p+1, 2 \cdot p] \), and will be carried out using Property 1. In all rounding modes the constants \( k_x = 10^{-x} \) used for this purpose, where \( x = q - p \), are pre-calculated to \( y \) bits as specified in Property 1. Since \( q \in [p+1, 2 \cdot p] \) for situations where rounding is necessary, all cases are covered correctly by choosing \( y = 1 + \lceil 2 \cdot p \rfloor \).

Similar to the case of the addition operation, the pre-calculated values \( k_x \) are stored for all \( x \in \{1, 2, \ldots, p\} \).

### Decimal Floating-Point Multiplication with Rounding to Nearest

The straightforward method to calculate the result is similar to that for addition. A new and better method for decimal floating-point multiplication with rounding to nearest that uses existing hardware for binary computations is presented in Algorithm 2. It uses Property 1 to avoid the need to calculate a remainder for the determination of midpoints or exact floating-point results, as shall be seen further. The multiplication algorithm has many similarities with the algorithm for addition.

**Algorithm 2.** Calculate the product of two decimal floating-point numbers rounded to nearest to \( p \) decimal digits, and determine its exactness.

\[
C' = C_1 \cdot C_2 \quad \text{// binary integer multiplication}
\]
\[
q = \text{the number of decimal digits required to}
\]
\[
\text{represent } C' \quad \text{// from table lookup}
\]
\[
\text{if } q \leq p \text{ then the result } n = C' \cdot 10^{e_1+e_2} \text{ is exact else}
\]
\[
\text{if } q \in [p+1, 2 \cdot p] \text{ continue}
\]

\[
x = q - p, \quad \text{the number of decimal digits to be removed from the lower part of } C', \ x \in [1, p]
\]
\[
C'' = C' + 1/2 \cdot 10^p \quad \text{// exact multiplication}
\]
\[
C'' = C' + 1/2 \cdot 10^p \quad \text{// pre-calculated}
\]
\[
k_x = 10^{-x} \quad (1 + \varepsilon), \ 0 < \varepsilon < 2^{-2 \cdot p + 1} \quad \text{// pre-calculated}
\]
\[
C' \approx C' \cdot k_x = C'' \cdot K_x \cdot 2^{\pm \varepsilon} \quad \text{// multiplication with implied binary point}
\]
^p is the fractional part of \( C^* \) // consists of the
\[
\text{if } 0 < f^* < 10^{-p} \text{ then } \quad \text{// } C^* = C'' \cdot K_x \cdot 2^{\pm \varepsilon},
\]
\[
\text{compare } E_\varepsilon \text{ bits shifted out of } C^* \text{ with } 0
\]
\[
\text{with } 10^p \quad \text{// and with } 10^p
\]
\[
\text{if } [C^*] \text{ is even then } C = [C^*] \quad \text{// logical right}
\]
\[
\text{shift; C has p decimal digits, correct by}
\]
\[
\text{// Property 1}
\]
\[
\text{else } C = [C^*] - 1 \quad \text{// if } [C^*] \text{ is odd // logical}
\]
\[
\text{right shift; C has p decimal digits, correct}
\]
\[
\text{// by Property 1}
\]
else
    C = \lfloor C^* \rfloor \text{ // logical shift right; } C \text{ has } p

n = C \cdot 10^{e1+e2+x} \text{ // rounding overflow}
if 0 < n < 10^p - 1/2 < 10^p \text{ then the result is exact }
else the result is inexact

// C* = C'' \cdot K_x \cdot 2^{-x} \Rightarrow compare E_x \text{ bits }
// shifted out of C* with 1/2 and 1/2+10^p

If q \geq p + 1 the result is inexact unless the x decimal digits removed from the lower part of C'' \cdot k_x were all zeros. To determine whether this was the case, just as for addition, the straightforward method is to calculate a remainder r = C' - C \cdot 10^x \in [0, 10^x). Midpoint results could be identified by comparing the remainder with 1/2 \cdot 10^x, and exact results by comparing the remainder with 0. However, the calculation of a remainder – a relatively costly operation – was avoided in Algorithm 2 and instead a single comparison to a pre-calculated constant was used.

The simplified method to determine midpoints and exactness along with the ability to use Property 1 make Algorithm 2 better for decimal floating-point multiplication than previously known methods.

### Decimal Floating-Point Multiplication when Rounding to Zero, Down, or Up

The method to calculate the result when rounding to zero or down is similar to that for rounding to nearest. Just as for addition, the step for calculating C''' = C' + 1/2 \cdot 10^x is not necessary anymore. Exactness is determined using the same method as in Algorithm 2. For rounding up, the calculation of the result and the determination of its exactness are identical to those for rounding down. However, when the result is inexact then one ulp has to be added to it.

### 6 DECIMAL FLOATING-POINT DIVISION

It will be assumed that the quotient

\[ n = (n_1 / n_2)_{\text{rdp}} = C \cdot 10^p \]

has to be calculated where n1 > 0, n2 > 0, and q1, q2, and q are the numbers of decimal digits needed to represent C1, C2, and C (the subscript rdp indicates rounding to p decimal digits, using the current rounding mode). Property 1 cannot be applied efficiently for the calculation of the result in this case because a very accurate approximation of the exact quotient is expensive to calculate. Instead, a combination of integer operations and floating-point division allow for the determination of the correctly rounded result. Property 1 is used only when an underflow is detected and the calculated quotient has to be shifted right a given number of decimal positions. The decimal floating-point division algorithm is based on Property 2 presented next.

**Property 2.** If a, b are two positive integers and m \in N, m \geq 1 such that b < 10^m, a/b < 10^m and n \geq \lceil m \log_2 10 \rceil, then |a/b - [(a)_{\text{mdp}}/(b)_{\text{mdp}}]| < 8.

The decimal floating-point division algorithm for operands n1 = C1 \cdot 10^{q1} and n2 = C2 \cdot 10^{q2} follows. While this algorithm may be rather difficult to follow without working out an example in parallel, it is included here for completeness. Its correctness, as well as that of all the other algorithms presented here has been verified.

**Algorithm 3.** Calculate the quotient of two decimal floating-point numbers, rounded to p decimal digits in any rounding mode, and determine its exactness.

if C1 \leq C2
find the integer d > 0 such that (C1/C2) \cdot 10^d \in [1, 10).
	// compute d based on the number
\text{ of decimal digits } q1, q2 \text{ in } C1, C2
\text{ } C1' = C1 \cdot 10^{d+15}, \text{ } Q = 0
\text{ } e = e1 - e2 - d - 15 // expected res. expon.
else\text{ } a = (C1 OR 1)_{\text{mdp}}, \text{ } b = (C2)_{\text{mdp}} \text{ // logical OR}
\text{ } Q = \lceil (a/b)_{\text{mdp}} \rceil
\text{ } Q2 = \lfloor (C1')_{\text{mdp}}/(C2')_{\text{mdp}} \rfloor
\text{ } Q = Q + Q2
if R < 0
\text{ } Q = Q - 1
\text{ } R = R + C2
if R = 0 the result n = Q \cdot 10^{16-d} is exact
else continue
\text{ } find the number of decimal digits for } Q:\text{ } d > 0 \text{ such that } Q \in [10^{d-1}, 10^d)
\text{ } C1' = R \cdot 10^{16-d}
\text{ } Q = Q \cdot 10^{16-d}
\text{ } e = e1 - e2 - 16 + d
\text{ } Q2 = \lfloor (C1')_{\text{mdp}}/(C2')_{\text{mdp}} \rfloor
\text{ } R = C1' - Q2 \cdot C2
\text{ } Q = Q + Q2
if R \geq 4 \cdot C2
\text{ } Q = Q + 4
\text{ } R = R - 4 \cdot C2
if R \geq 2 \cdot C2
\text{ } Q = Q + 2
\text{ } R = R - 2 \cdot C2
if R \geq C2
\text{ } Q = Q + 1
\text{ } R = R - C2
if e \geq \text{minimum decimal exponent}
\text{ } apply rounding in desired mode by
comparing R and C2
// e.g. for rounding to nearest add 1 to Q
// if 5 ⋅ C2 < 10 ⋅ R + (Q AND 1)
the result n = Q ⋅ 10^e is inexact
else
result underflows
compute the correct result based on Prop. 1

7 DECIMAL FLOATING-POINT SQUARE ROOT

Assume that the square root
n = (√n)_{rnd,p} = C ⋅ 10^e
has to be calculated (where the subscript rnd,p indicates rounding to p decimal digits using the current rounding mode). The method used for this computation is based on Property 3 and Property 4, shown next. A combination of integer and floating-point operations are used. It will be shown next that the minimum precision n of the binary floating-point numbers that have to be used in the computation of the decimal square root for decimal64 arguments (with p = 16) is n = 53, so the double precision floating-point format can be used. The minimum precision n of the binary floating-point numbers that have to be used in the computation of the square root for decimal128 arguments (with p = 34) is n = 113, so the quad precision floating-point format can be used safely.

Properties 3 and 4 as well as the algorithm for square root calculation are included here for completeness.

Property 3. If x ∈ (1, 4) is a binary floating-point number with precision n and s = (√x)_{RN,n}
is its square root rounded to nearest to n bits, then s + 2^−n < x.

Property 4. Let m be a positive integer and n = \left\lceil m \log_2 10 + 0.5 \right\rceil. For any integer C ∈ [10^{m-1}, 10^m), the inequality |\sqrt{C} − \lfloor (C)_{RN,n} \rfloor| < 3/2 is true.

The round-to-nearest decimal square root algorithm can now be summarized as follows:

Algorithm 4. Calculate the square root of a decimal floating-point number n1 = C ⋅ 10^e, rounded to nearest to p decimal digits, and determine its exactness.
if e is odd then
  e′ = e − 1
  C′ = C ⋅ 10
else
e′ = e
C′ = C
let S = \lfloor (C')_{RN,n} \rfloor
if S ⋅ S = C'
the result n = S ⋅ 10^{e'/2} is exact
else
  q = number of decimal digits in C
  C'' = C’ ⋅ 10^{p−1−q} and Q = \lfloor (C’')_{RN,n} \rfloor
  if (C’’ − Q ⋅ Q < 0) sign = −1 else sign = 1
  M = 2 ⋅ Q + sign // will check against this
  // midpoint for rounding to nearest
  if (M ⋅ M − 4 ⋅ C’’ < 0) sign_m = −1
  else sign_m = 1
  if sign ≠ sign_m Q’ = Q + sign else Q’ = Q
the result n = Q’ ⋅ 10^{e'/2} is inexact

8 CONCLUSIONS

A new generic implementation in C of the basic operations for decimal floating-point arithmetic specified in the IEEE 754R standard proposal was completed, based on new algorithms presented in this paper. Several other operations were implemented that were not discussed here for example remainder, fused multiply-add, comparison, and various conversion operations. Performance results for all basic operations were in the expected range, for example the latency of decimal128 operations is comparable to that of binary quad precision operations implemented in software. It was also possible to compare the performance of the new software package for basic operations with that of the decNumber package contributed to GCC (Grimm, 2005). The decNumber package represents the only other implementation of the IEEE 754R decimal floating-point arithmetic in existence at the present time. It should be noted that decNumber is a more general decimal arithmetic library in ANSI C, suitable for commercial and human-oriented applications (decNumber, 2005). It allows for integer, fixed-point, and decimal floating-point computations, and supports arbitrary precision values (up to a billion digits).
Tests comparing the new decimal floating-point library using the algorithms described in this paper versus decNumber showed that the new generic C implementations for addition, multiplication,
division, square root, and other operations were faster than the decNumber implementations, in most cases by one to two orders of magnitude.

Table 2 shows the results of this comparison for basic 64-bit and 128-bit decimal floating-point operations measured on a 3.4 GHz Intel® EM64t system with 4 GB of RAM, running Microsoft Windows Server 2003 Enterprise x64 Edition SP1. The code was compiled with the Intel(R) C++ Compiler for Intel(R) EM64T-based applications, Version 9.0. The three values presented in each case represent minimum, median, and maximum values for a small data set covering operations from very simple (e.g. with operands equal to 0 or 1) to more complicated, e.g. on operands with 34 decimal digits in the 128-bit cases. For the new library, further performance improvements can be attained by fine-tuning critical code sequences or by optimizing simple, common cases.

Table 2: New Decimal Floating-Point Library Performance vs. decNumber on EM64t (3.4 GHz Xeon).
Minimum-median-maximum values are listed in sequence, after subtracting the call overhead.

<table>
<thead>
<tr>
<th>Operation</th>
<th>New Library [clock cycles]</th>
<th>decNumber Library [clock cycles]</th>
<th>dec Number /New Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>64-bit ADD</td>
<td>14-140-241</td>
<td>99-1400-1741</td>
<td>4-10-14</td>
</tr>
<tr>
<td>64-bit MUL</td>
<td>21-120-215</td>
<td>190-930-1824</td>
<td>6-8-9</td>
</tr>
<tr>
<td>64-bit DIV</td>
<td>172-330-491</td>
<td>673-2100-3590</td>
<td>4-6-11</td>
</tr>
<tr>
<td>64-bit SQRT</td>
<td>15-288-289</td>
<td>82-16700-18730</td>
<td>7-58-107</td>
</tr>
<tr>
<td>128-bit ADD</td>
<td>16-170-379</td>
<td>97-2300-3333</td>
<td>4-13-14</td>
</tr>
<tr>
<td>128-bit MUL</td>
<td>19-300-758</td>
<td>95-3000-4206</td>
<td>5-10-18</td>
</tr>
<tr>
<td>128-bit DIV</td>
<td>153-250-1049</td>
<td>1056-2000-7340</td>
<td>4-8-9</td>
</tr>
<tr>
<td>128-bit SQRT</td>
<td>16-700-753</td>
<td>61-42000-51855</td>
<td>4-60-152</td>
</tr>
</tbody>
</table>

For example for the 64-bit addition operation the new implementation, using the 754R binary encoding for decimal floating-point, took between 14 and 241 clock cycles per operation, with a median value around 140 clock cycles. For the same operand values decNumber, using the 754R decimal encoding, took between 99 and 1741 clock cycles, with a median around 1400 clock cycles. The ratio shown in the last column was between 4 and 14, with a median of around 10 (probably the most important of the three values).

It is also likely that properties and algorithms presented here for decimal floating-point arithmetic can be applied as well for a hardware implementation, with re-use of existing circuitry for binary operations. It is the authors’ hope that the work described here will represent a step forward toward reliable and efficient implementations of the IEEE 754R decimal floating-point arithmetic.

REFERENCES
M. Erle, E, Schwarz, and M. Schulte, Decimal Multiplication with Efficient Partial Product Generation, 17th Symposium on Computer Arithmetic, 2005
Keywords: Process types, synchronization, type systems, race-free programs.

Abstract: Process types – a kind of behavioral types – specify constraints on message acceptance for the purpose of synchronization and to determine object usage and component behavior in object-oriented languages. So far process types have been regarded as a purely static concept for Actor languages incompatible with inherently dynamic programming techniques. We propose solutions of related problems causing the approach to become usable in more conventional dynamic and concurrent languages. The proposed approach can ensure message acceptability and support local and static checking of race-free programs.

1 INTRODUCTION

Process types (Puntigam, 1997) represent a behavioral counterpart to conventional object types: They support subtyping, genericity, and separate compilation as conventional types. Additionally they specify abstractions of object behavior. Abstract behavior specifications are especially desirable for software components, and they can be used for synchronization. Both concurrent and component-based programming are quickly becoming mainstream programming practices, and we expect concepts like process types to be important in the near future. However, so far process types are not usable in mainstream languages:

1. Their basis are active objects communicating by message passing (Agha et al., 1992). Variables are accessible only within single threads. In mainstream languages like Java, threads communicate through shared (instance) variables; one thread reads values written by another. To support such languages we must extend process types with support of shared variables.

2. Process types are static. Object state changes must be anticipated at compilation time. We must adapt process types to support dynamic languages like Smalltalk (using dynamic process type checking).

Support of dynamic languages turns out to be a good basis for supporting communication through shared variables. Hence, we address mainly the second issue and show how dynamic type checking can deal with the first issue.

We introduce the basic static concept of process types for a conventional (Java-like) object model in Section 2. Then, we add support of dynamic synchronization in Section 3 and of shared variables with late type checking in Section 4. Local and static checking of race-free programs is rather easy in our setting as discussed in Section 5.

2 STATIC PROCESS TYPES

Figure 1 shows the grammar of TL1 (Token Language 1) – a simple Java-like language we use as showcase. We differentiate between classes and types without implementations. To create a new object we invoke a creator new in a class. Type annotations follow after "": Token declarations (names following the keyword token), tokens occurring within square brackets in types, and with-clauses together determine the statically specified object behavior.

The first example shows how tokens allow us to specify constraints on the acceptability of messages:

```
type Buffer is
    token empty filled
    put(e:E with empty->filled)
    get(with filled->empty): E
```

According to the with-clause in put we can invoke put only if we have an empty; this token is removed.
on invocation, and filled is added on return. For
x of type Buffer[empty] – a buffer with a single
token empty – we invoke x.put(..). This invo-
cation changes the type of x to Buffer[filled].
Next we invoke x.get(), then x.put(.,.), and
so on. Static type checking enforces put and get to
be invoked in instances of Buffer[empty] in al-
ternation. Type checking is simple because we need
only compare available tokens with tokens required
by with-clauses and change tokens as specified by
with-clauses (Puntigam, 1997).

The type Buffer[empty,8 filled,7] de-
notes a buffer with at least 8 filled and 7 empty slots.
An instance accepts put and get in all sequences
such that the buffer never contains more that 15 or
less than zero elements as far as the client knows.

In the next example we show how to handle tokens
in parameter types similarly as in with-clauses:

class Test is
play(b:Buffer[filled->filled])
do e:E = b.get() -- b:Buffer[empty]
e = e.subst() -- another e
b.put(e)
copy(b:Buffer[empty,filled->filled,2])
do e:E = b.get() -- b:Buffer[empty,2]
b.put(e) -- b:Buffer[empty,2]
b.put(e) -- b:Buffer[filled,2]

Let y be of type Buffer[empty,2 filled,2]
and x of type Test. We can invoke x.play(y)
since y has the required token filled. This rou-
tine gets an element from the buffer, assigns it to the
local variable e (declared in the first statement), as-
signs a different element to e, and puts this element
into the buffer. Within play the buffer is known
to have a single filled slot on invocation and on ret-
urn. For the type of b specified in the formal pa-
rameter list it does not matter that the buffer has been
empty meanwhile and the buffer contents changed.

After return from play variable y is still of type
Buffer[empty,2 filled,2].

Invocations of copy change argument types: On
return from x.copy(y) variable y will be of type
Buffer[empty, filled,3]. Removing tokens
to the left of -> on invocation causes the type to be-
become Buffer[empty, filled], and adding the
tokens to the right on return causes it to become
Buffer[empty, filled,3].

Parameter passing does not produce or consume to-

dents. Tokens just move from the argument type to the
parameter type on invocation and vice versa on return.

Only with-clauses can actually add tokens to and re-
move them from an object system. This is a basic
principle behind the idea of tokens: Each object can
produce and consume only its own tokens.

A statement ‘fork x.copy(y)’ spawns a new
thread executing x.copy(y). Since the execution
continues without waiting for the new threads, in-
voiced routines cannot return tokens. The type of y
changes from Buffer[empty,2 filled,2] to
Buffer[empty, filled,3]. The type of y is split
into two types – the new type of y and the type of b.
Both threads invoke routines in the same buffer with-
out affecting each other concerning type informa-
tion.

Assignment resembles parameter passing in the
case of spawning threads: We split the type of an
assigned value into two types such that one of the
split types equals the current static type of the vari-
able, and the remaining type becomes the new type of
the assigned value. Thereby, tokens move from the
value’s to the variable’s type. If the statically evalua-
ted type of v is Buffer[empty,2] and y is of
type Buffer[empty,2 filled,2], then v=y
causes y’s type to become Buffer[filled,2].

Local variables are visible in just a single thread of
control. This property is important because it allows
us to perform efficient type checking by a single walk
through the code although variable types can change
with each invocation. Because of explicit formal pa-
rameter types we can check each class separately. If
variables with tokens in their types were accessible
in several threads, then we must consider myriads of
possible interleavings causing static type checking to
become practically impossible. Instance variables can
be shared by several threads. To support instance vari-
ables and still keep the efficiency of type checking we
require their types to carry no token information. We
address this restriction in Section 4.

Explicit result types of creators play a quite im-
portant role for introducing tokens into the system:
class Buffer1 < Buffer is
s:E = new() 
put(e:E with empty->filled) do s=e
get(with filled->empty):E do return s
new(): Buffer1[empty] do null

Class Buffer1 inherits empty and filled from
The creator introduces just a single token sync. Different from TL2 just by missing type annotations on for-clauses in creators. Tokens to the left of \( \Rightarrow \) in when-clauses must be available and are removed before executing the body, and tokens to the right are added on return. Different from with-clauses, when-clauses require dynamic tokens to be in the object’s token set and change this token set. If required dynamic tokens are not available, then the execution is blocked until they become available. Checks for token availability occur only at run time. The following variant of the buffer example uses static tokens to avoid buffer overflow and underflow, and dynamic tokens to ensure mutual exclusion:

```java
class Buffer50 < Buffer is
ten sync
    lst: List
    new(): Buffer50[empty.50] \Rightarrow sync do
        lst = List.new()
    put(e:E with empty->filled)
        when sync->sync do lst.addLast(e)
    get(with filled->empty): E
        when sync->sync do
            return lst.getAndDeleteFirst()

The creator introduces just a single token sync. Both put and get remove this token at the begin and issue a new one on return. Clients need not know about the mutual exclusion of all buffer operations. Of course we could use only dynamic tokens which is more common and provides easier handling of buffers.

Static and dynamic tokens live in mostly independent worlds. Nonetheless we have possibilities to move tokens from the static to the dynamic world and vice versa as shown in the following example:

class StaticAndDynamic is
ten t
    beDynamic(with \( \Rightarrow \) t) when \( \Rightarrow \) t do null
    beStatic(with \( \Rightarrow \) t) when \( \Rightarrow \) t do null
    new(): StaticAndDynamic[t] do null

There always exists only a single token \( t \) for each instance, no matter how often and from how many threads we invoke beDynamic and beStatic.

The major advantage of our approach compared to concepts like semaphores and monitors is the higher level of abstraction. It is not so easy to “forget” to release a lock as often occurs with semaphores, and it is not necessary to handle wait queues using wait and notify commands as with monitors. For static tokens we need not execute any specific synchronization code at all. This synchronization is implicit in the control flow.

4 DYNAMIC TYPING

In TL1 and TL2 we constrained the flexibility of the language to get efficient static type checking. Types of instance variables cannot carry tokens. In this section we take the position that static type checking is no precondition for the token concept to be useful. We want to increase the language’s flexibility (by supporting tokens on instance variables) and nonetheless ensure that synchronization conditions expressed in with-clauses are always satisfied. An error shall be reported before invocations if required tokens are not available.

Figure 3 shows the grammar of TL3 that differs from TL2 just by missing type annotations on formal parameters and declarations. However, without type annotations there is no explicit information about available tokens. We handle this information dynamically. One kind of type annotation is left in TL3: Types of new instances returned by creators must be specified explicitly because tokens in this type to-take with with-clauses determine which routines can be invoked. Such types are part of behavior specifications. Except of type annotations the following example in TL3 equals Buffer50:
The variable win etc. are not supported. Class WManager is

```plaintext
type BufferDyn is
  token empty filled
  put(e with empty->filled)
  get(with filled->empty)

class Buffer50Dyn < BufferDyn is
  token sync
  lst::Buffer50Dyn[empty.50]->sync do
    new():Buffer50Dyn[empty.50]->sync do
      lst = List.new()
      put(e with empty->filled)
      when sync->sync do lst.addLast(e)
      get(with filled->empty)
      when sync->sync do
        return lst.getAndDeleteFirst()

  end

end
```

class WindowImpl < Window is

```plaintext
new(): WindowImpl[closed] do ...
...
```

class WManager is

```plaintext
win::
  new(w):WManager do win=w win.setup()
  onButton1() do win.iconify()
  onButton2() do win.close()
  onButton3() do win.display()
```

Some state changes (directly from an icon to closed, etc.) are not supported. Class WManager specifies actions to be performed when users press buttons. Under the assumption that a displayed window has only Button 1 and 2 and an icon only Button 3 the constraints on state changes are obviously satisfied. Since the assumption corresponds to the existence of at most one token for each window we need nothing else to ensure a race-free program. We express the assumption by with-clauses and dynamically ensure them to be satisfied. The variable win must be associated with a (static) token specifying the window’s state. In TL1 and TL2 we cannot express such type information that is implicit in TL3.

TL3 deals with dynamic tokens in the same way as TL2. To dynamically handle information about available static tokens we consider two approaches – TL3flex as a simple and flexible approach, and TL3strict as a more restrictive and safer approach.

**TL3flex.** In TL3flex we tread static tokens in a similar way as dynamic tokens: Each objects contains a pool of static tokens. On invocations tokens to the left of -> in with-clauses are taken from the pool, and on return those to the right are added to the pool. An error is reported if the pool does not contain all required tokens.

This approach is very flexible. Each thread can use all previously issued static tokens no matter which thread caused the tokens to be issued. A disadvantage is a low quality of error messages because there is no information about the control flow causing tokens not to be available. Furthermore, there is a high probability for program runs not to uncover synchronization problems. Thus, program testing is an issue.

**TL3strict.** To improve error messages and the probability of detecting problems we dynamically simulate static type checking: Instead of storing static tokens centralized in the object we distribute them among all references to the object. On invocation we check and update only tokens associated with the corresponding reference. We must find an appropriate distribution of tokens among references. In TL1 and TL2 the programmer had to determine the distribution by giving type annotations. In TL3strict we distribute tokens lazily as needed in the computation.

Instead of splitting a token set on parameter passing or assignment we associate the two references with pointers to the (unsplit) token set as well as with a new empty token set for each of the two references. Whenever required tokens are not available in the (after assignment or parameter passing empty) token set of a reference we follow the pointers and take the tokens where we find them. New tokens are stored in the references’ own token sets. This way all references get the tokens they need (if available) and we need not foresee how to split token sets. Repeated application leads to a tree of token sets with pointers from the leaves (= active references) toward the root (= token set returned by creator). We report an error only if tokens required at a leaf cannot be collected from all token sets on the path to the root. On return of invocations we let actual parameters point to token sets of corresponding formal parameters.

Figure 4 shows an example: Immediately after creating a window there is only one reference n to it (a). The box contains the single token in the corresponding token set. When invoking new in WManager using n as actual parameter we construct new token sets for n and for the formal parameter w (b). When the creator assigns w to win we add new token sets for w and win (c). An invocation of setup on win removes the token closed and adds displ. On return from the creator we let the token set of n point to that of w (d). Now only win carries the single token. We cannot change the window’s state through n. Therefore, TL3strict is safer and less flexible than TL3flex.

We can build large parts of the structures shown in Figure 4 already at compilation time by means of ab-
The algorithm is accurate in the sense that treatment the algorithm always reaches a fixed point. If a token set differs from the token set from which it is obtained, then we remove the smaller token set. And the token set contains all tokens occurring in another token set. If a token set returned by the algorithm does not contain \( \infty \), then there exists a sequence of invocations producing exactly this token set, and if a token set returned by the algorithm contains \( \infty \), then there exist invocation sequences producing corresponding tokens without upper bounds.

In TL2 and TL3 we must consider static and dynamic tokens together to get most accurate results. Since the static and the dynamic world are clearly separated, static and dynamic tokens must not be intermixed. We have to clearly mark each token as either static or dynamic (for example, by an index) and regard differently marked tokens as different. The algorithm starts with one token set for each creator containing both static and dynamic tokens. A new token set is constructed by simultaneously removing and adding tokens as specified in the with- and when-clause of a routine. The result shows which dynamic tokens can exist together with static tokens. For example, applied to StaticAndDynamic (see Section 3) the algorithm returns two token sets, one containing only a static token \( \tau \) and the other only a dynamic token \( \tau \); in this case no dynamic token can exist at the same time as a static one.

Once we know the upper bounds it is easy to perform our check of race-free programs as shown in the following pseudo-code:

```plaintext
let U be the upper-bound set of token sets of class c;
for each instance variable of c
for each routine r write-accessing v
    for each routine s (read or write) accessing v
        let p be the union of the token sets to the left of \( \rightarrow \) in r and s;
        if there is a \( u \in U \) containing all tokens in p then issue a warning about a potential race;
    otherwise c is race-free.
```

As an example we apply this check to Buffer1 (see Section 2). As upper-bound set of token sets \( S \) we have \{\{empty\},\{filled\}\}; there is always at most one token empty or filled. The only instance variable \( s \) is written in put and read in get. Hence, \( r \) ranges just over put, \( s \) over put and get, and \( p \) over \{empty,2\} and \{empty, filled\}. The class is race-free because no token set in \( S \) contains two empty or an empty and a filled.

The set \( S \) can become quite large because of combinatorial explosion. For example, \( S \) constructed for Buffer50Dyn contains 51 different token sets—all possibilities of summing up tokens of two names to 50 tokens. Fortunately, a simple change in the algorithm to compute upper bounds can reduce the size of \( S \) considerably: When computing the fixed point we replace all token numbers larger than \( 2 \cdot n^2 \cdot i \) by \( \infty \), where \( n \) is the largest total number of tokens to the left of \( \rightarrow \) in the with- and when-clause of the same routine, and \( i \) is the number of different token names.
in the class. For Buffer50Dyn we have \( n = 2, i = 3, 2 \cdot n^2 \cdot i = 24 \), and \( S \) contains just a single token set \{sync, empty, \infty, filled, \infty\}. This optimization does not change the output of the race-freeness check: Soundness is not affected because the multi-set of supposedly reachable tokens in a system can just get larger. No token set \( p \) (as in the algorithm) can contain more than \( 2 \cdot n \) tokens, and a single token of some name can be generated from no more than \( n \cdot i \) tokens of another name. Therefore, more than \( 2 \cdot n^2 \cdot i \) tokens of one name can be ignored for our purpose. Probably there are more accurate estimations, but we expect this simple one to be sufficient because token numbers to the left of \( \rightarrow \) are usually small.

All information needed to check race-free classes is explicit in TL1, TL2, and TL3. We need no information about formal parameter types and no aliasing information. No global program analysis is necessary.

6 DISCUSSION, RELATED WORK

The idea of integrating process types into dynamic languages is new and at a first glance unexpected because such types were developed to move dynamic aspects like synchronization to the static language level whenever possible (Puntigam, 1995; Puntigam, 1997; Puntigam, 2000). In some sense the integration of more advanced static concepts into dynamic languages is a consistent further development allowing us to use the appropriate (static or dynamic) concept for each task. Such integration helps us to deepen our understanding of related concepts.

We usually regard synchronization of concurrent threads as a purely dynamic concept: If there is a dependence between two control flows, then one of the corresponding threads must wait until the other thread has caught up to meet the synchronization point. Since threads usually run asynchronously and at statically unpredictable speed, it is only possible to decide at run time whether a thread must wait at a synchronization point. However, these considerations are valid only at a very low level (close to the hardware) point of view. From the programmers’ higher level point of view it is quite often not clear whether there exist dependences between threads or not. Using explicit synchronization as with monitors, semaphores, rendezvous communication, etc. programmers must add much more synchronization points than are actually necessary. There are optimization techniques that can statically eliminate up to about 90% (about 60% in average) of all locks from Java programs and thereby considerably improve program performance (von Praun and Gross, 2003). Probably even more synchronization points are actually not necessary.

Current programming languages allow programmers to write programs with races although there are many proposals to ensure race-free programs (Bacon et al., 2000; Boyapati and Rinard, 2001; Brinch-Hansen, 1975; Flanagan and Abadi, 1999). Applications of such techniques may lead to further increase of unnecessary synchronization because no approach can accurately decide between necessary and unnecessary locks. Nonetheless, these techniques are very useful because races are an important practical problem in concurrent programming.

Process types were developed as abstractions over expressions in process calculi (Puntigam, 1995). These abstractions specify acceptable messages of active objects and allow the acceptability to change over time (thereby specifying synchronization constraints). Static type checking ensures that only acceptable messages can be sent and enforces all synchronization constraints to be satisfied. In this sense type checking in process types has a similar purpose as ensuring race-free programs. However, process types allow us to specify arbitrary constraints on message acceptability, not just synchronization necessary to avoid races. In fact, the underlying calculi do not support shared data that could suffer from races.

There is a clear tendency toward more and more complex interface specifications going far beyond simple signatures of available routines (Arbab, 2005; de Alfaro and Henzinger, 2001; Heuzeroth and Reussner, 1999; Jacobsen and Krämer, 1998; Lee and Xiong, 2004; Mezini and Ostermann, 2002; Nierstrasz, 1993; Plasil and Visnovsky, 2002; Yellin and Strom, 1997). We consider such interfaces to be partial specifications of object behavior (Liskov and Wing, 1993). They are especially valuable to specify the behavior of software components as far as needed for component composition. Process types are useful as partial behavior specifications (Puntigam, 2003; Südholt, 2005). We regard behavior specifications as the major reason for using process types.

Pre- and postconditions in with-clauses allow us to specify a kind of contracts between components (Meyer, 1997; Meyer, 2003). Such contracts clearly specify responsibilities of software and help us to move responsibilities from one component to another. For example, we move the responsibility of proper synchronization from the server to the client if we use with-clauses instead of when-clauses.

Behavioral types and synchronization of concurrent threads are related topics: Specifications of object behavior cannot ignore necessary synchronization if we expect components composed according to their behavioral types to work together in concurrent environments, and constraints on message acceptability specify a kind of synchronization. The present work allows programmers to decide between synchronization globally visible through the interface (with-clauses) and local synchronization regarded as an im-
7 CONCLUSION

Behavioral types like process types gain more and more importance especially together with component composition. By partially specifying object behavior these types express synchronization in the form of software contracts clearly determining who is responsible for proper synchronization. Process types use simple token sets as abstractions over object states.

In this paper we explored how to add process types to rather conventional object-oriented programming languages. As a showcase we developed the languages TL1 to TL3. Static type checking in TL1 ensures that all conditions in with-clauses are satisfied, this is, all required tokens are available. We can synchronize concurrent threads just by waiting for messages. To overcome the restriction, TL2 adds a new dynamic concept of synchronization based on token sets. Neither TL1 nor TL2 can deal with static token sets associated with instance variables because of possible simultaneous accesses by concurrent threads. In TL3 we dispense with static types and apply one of two methods to dynamically ensure the availability of required tokens – a flexible method and one with better error messages and partial support of static type checking. All variables in TL3 have only dynamic types that can implicitly carry tokens. In the three languages we can ensure race-free programs by checking each class separately, without any need of global aliasing information.

Our approach uses token sets for several related purposes – synchronization of concurrent threads and statically and dynamically checked abstract behavior specifications. It is a major achievement to integrate these concepts because of complicated interrelations. The integration is valuable because it gives software developers much freedom and at the same time clear contracts and type safety.

Much work on this topic remains to be done. For example, currently our algorithm can issue warnings about potential races even in purely sequential program parts. Many other approaches to ensure race-free programs put much effort into detecting sequential program parts. By integrating such approaches into our algorithm we expect to considerably improve the accuracy. Most approaches to remove unnecessary locking from concurrent programs also work on sequential program parts (Choi et al., 1999; von Praun and Gross, 2003; Vivien and Rinard, 2001). We expect a combination of the techniques to improve runtime efficiency.

REFERENCES


ASPECTBOXES – CONTROLLING THE VISIBILITY OF ASPECTS

Alexandre Bergel¹, Robert Hirschfeld², Siobhán Clarke¹ and Pascal Costanza³

¹ Distributed Systems Group
Trinity College Dublin, Ireland
{Alexandre.Bergel, Siobhan.Clarke}@cs.tcd.ie

² Hasso-Plattner-Institut
Universität Potsdam, Germany
hirschfeld@hpi.uni-potsdam.de

³ Programming Technology Lab
Vrije Universiteit Brussel, Belgium
pascal.costanza@vub.ac.be

Keywords: Aspect-oriented programming, aspect composition, scoping change, aspects, classboxes, squeak.

Abstract: Aspect composition is still a hot research topic where there is no consensus on how to express where and when aspects have to be composed into a base system. In this paper we present a modular construct for aspects, called aspectboxes, that enables aspects application to be limited to a well defined scope. An aspectbox encapsulates class and aspect definitions. Classes can be imported into an aspectbox defining a base system to which aspects may then be applied. Refinements and instrumentation defined by an aspect are visible only within this particular aspectbox leaving other parts of the system unaffected.

1 INTRODUCTION

Aspect-oriented programming (AOP) promises to improve the modularity of programs by providing a modularity construct called aspect to clearly and concisely capture the implementation of crosscutting behavior. An aspect instruments a base software system by inserting pieces of code called advices at locations designed by a set of pointcuts.

An important focus of current research in AOP is on aspect composition (Douence et al., 2004; Klaeren et al., 2000; Nagy et al., 2005; Brichau et al., 2002). Ordering and nesting are commonly used when composing aspects and advices (Kiczales et al., 2001; Tanter, 2006). Whereas most aspect languages provide means to compose aspects at a very fine grained level, experience has shown that ensuring a sound combination of aspects is a challenging and difficult task (Lopez-Herrejon et al., 2006). First steps are already taken by AspectJ (Kiczales et al., 2001) by restricting pointcuts to a Java package or a class through the use of dedicated pointcuts primitives such as within and withincode primitive pointcuts.

If we regard an aspect as an extension to a base system, multiple extensions are difficult to manage and control, even if they are not interacting with each other. We believe that the reason for this is the lack of a proper scoping mechanism.

In this paper we define a new modular construct for an aspect language called an aspectbox. An aspectbox is a modular unit that may contain class and aspect definitions. Classes can be imported into an aspectbox and the aspect is then applied to the imported classes. Refinements originated from such aspects are visible only within the aspectbox that defines this aspect. Outside this aspectbox the base system behaves as if there were no aspect. Other parts outside a particular aspectbox remain unaffected.

In Section 2 we provide an example illustrating the issues when composing aspects. In Section 4 we describe the aspectboxes module system and its properties. In Section 5 we present our Squeak-based implementation of aspectboxes. Related work is discussed in Section 6. We conclude by summarizing the presented work in Section 7.

2 MOTIVATION

To motivate the need for limiting the scope of aspects, we use an example based on the design of a small four-wheel electric car, and its implementation based on a mainstream aspect language, AspectJ (Kiczales et al., 2001).

The CyCab (Baille et al., 1999) is an electric four wheel car designed to transport up to two people. The
mechanics is taken from a small electrical golf car frame. Functionalities implemented in a CyCab range from an autonomous driving facility (like a coach in a train) to ultrasonic sensors for collision avoidance. A CyCab is composed of three different units (driving control, position control and safety control). Each unit is composed of one or more modules. Figure 1 illustrates the architecture of a CyCab.

Driving Control. A CyCab is steered with a joystick emitting electrical pulses used by the motion engine to activate the four motored wheels. This feature is provided by three modules within the driving control unit. The joystick module emits signals that are captured by the motion engine module. This module controls the wheels.

Position Control. The position control unit computes the velocity and the location of the CyCab based on the acceleration given by the motion engine to the wheels and their angle between the car head.

Safety Control. The safety control unit verifies the interactions between the three modules of the driving control unit. For example, it asserts that pulses emitted by the joystick trigger the correct reaction in the engine and the wheels reflect the heading dictated by the joystick. In addition, in the event of failure the power is shut down and communication between the three modules is cut off.

Figure 1: The three units and their modules that compose the CyCab electrical car.

3 EXAMPLE ANALYSIS

Behavior defined by the safety control unit crosscuts the whole driving control unit. For example, the impact of a power shut-down is that the joystick, the motion engine and wheels are disconnected. This can be easily captured in an aspect that adds behaviour to check the power status into each affected module, as implemented by the following AspectJ aspect:

```java
aspect PowerOff {
    private boolean hasPower = ...;
    pointcut drivingControl():
        target(Joystick) && call(public * *(..)) ||
        target(Engine) && call(public * *(..)) ||
        target(Wheels) && call(public * *(..));
    void around(): drivingControl()
        if (hasPower == true)
            proceed();
    ...
}
```

The PowerOffAndBreak aspect inserts a check before all public methods of the classes Joystick, Engine and Wheels to proceed only if power is equal to true. This aspect is applied to the driving control unit and has to be composed with the PositionAndVelocity aspect defined by the position control unit:

```java
aspect PositionAndVelocity {
    double speed;
    pointcut speedUp() : call (* Engine.accelerate());
    after(): speedUp() {
        //... Speed calculation
    }
    ...
}
```

PositionAndVelocity inserts a speed calculation functionality after the execution of the accelerate method. Defining the position and velocity module as an aspect has the benefit to leave the driving control unit free from referring to the speed and position computation. The two aspects PowerOffAndBreak and PositionAndVelocity are woven into the base system, the driving control unit, to form a deployable system. With current aspect languages such as AspectJ, extensions defined by all aspects are automatically applied to all the modules in the system (i.e., the physical display screen, the electronic control unit in charge of the safety).

This facility is particularly dangerous regarding the implicit sharing of the control flow of the application. A failure raised by the PositionAndVelocity aspect may easily impact the PowerOffAndBreak aspect affecting the electronic control unit in charge of the safety.

Whereas most of current aspect languages offer sophisticated pointcut primitives to express location of join points, they do not provide a means to limit the impact of an aspect into a well-defined system area.
In the upcoming section we define a module system for an aspect-oriented programming environment in which one or more aspect compositions are effective only in the context of a well-defined subset of the base system.

4 SCOPING ASPECTS WITH ASPECTBOXES

Most of today’s aspect languages do not provide a way to limit the impact of an aspect within a delimited scope. In this section, we describe a module system for an aspect-oriented programming language that allows for controlling the visibility of a set of aspects relative to a well-defined system area.

4.1 Aspectboxes in a Nutshell

Aspectboxes is a namespace mechanism for aspects. An aspect lives in an aspectbox and the effects of this aspect is limited to the aspectbox in which it is defined and to other aspectboxes that rely on the base system extended by this aspect. An aspectbox can (i) define classes, (ii) import classes from another aspectbox and (iii) define aspects.

The import relationship is transitive: If an aspectbox AB2 imports a class C from another aspectbox AB1, then a third aspectbox AB3 can import C from AB2. From the point of view of the importing aspectbox AB3, there is no difference if the class is defined or imported in the provider aspectbox AB2. Because aspects cannot be reused across multiple base systems, aspects cannot be imported.

A pointcut definition contained in an aspect refers only to classes that are imported (i.e., visible within the aspectbox that defines this aspect). An aspect in an aspectbox refines the behavior of the classes that are imported or defined, for instance by adding some code before and after some methods. The classes augmented with the aspect can also be imported from another aspectbox. From the point of view of an importing aspectbox, there is no distinction between classes defined within the aspectbox and those imported.

4.2 Namespace for Classes and Aspects

An aspectbox defines a namespace for class definitions, aspect definitions and aspect compositions.

Aspectbox as namespace for classes. The class Engine contained in the aspectbox DrivingControlAB as illustrated in Figure 1 is defined as the following:

(Aspectbox named: #DrivingControlAB)
createClassNamed: #Engine
instanceVariableNames: "

The class Engine does not have any instance variables and two methods accelerateWheels: anAcceleration and setAnglewithHeading: anAngle are defined on it.

DrivingControlAB.Engine>>
 accelerateWheels: anAcceleration
 "accelerate the wheels with a given acceleration"
...

DrivingControlAB.Engine>>
 setAnglewithHeading: anAngle
 "set the heading of the car by setting appropriately the wheel angle"
...

An aspectbox acts as a code packaging mechanism and constrains aspect visibility. A class is visible within an aspectbox if this class is defined in or imported to this aspectbox. Any class visible within an aspectbox AB1 can be imported from AB1 by other aspectboxes. The aspectbox PositionControlAB imports the class Engine from DrivingControlAB

(Aspectbox named: #PositionControlAB)
import: #Engine from: #DrivingControlAB

An instantiation of a class can occur in any aspectbox as long as this class is visible in the aspectbox that contains the code performing the instantiation. Class instances (i.e., objects) do not belong to an aspectbox.

Aspectbox as namespace for aspect definitions.
The module position and velocity is implemented by the PositionAndVelocity aspect:

(Aspectbox named: #PositionControlAB)
createAspectNamed: #PositionAndVelocity
instanceVariableNames: 'heading velocity'

Because the aspect PositionAndVelocity has to be applied to the class Engine, this class has to be imported from the DrivingControlAB aspectbox. This aspect also defines advices to be applied to the methods accelerateWheels: anAcceleration and setAnglewithHeading: anAngle that compute the velocity and the heading, respectively, as illustrated in Figure 2.

Aspectbox as namespace for aspect compositions.
An aspect, which is defined in an aspectbox, is applied to classes that are visible in this aspectbox (i.e., classes that are imported or defined). The effect of this aspect is limited to the aspectbox in which this

---

1We end the name of aspectboxes by AB to clearly make a distinction between them and regular class names.

2Since our aspectboxes prototype is implemented in Squeak, we therefore use the Squeak syntax to describe them.
PositionControlAB. PositionAndVelocity>> adviceComputeVelocity

"AfterAdvice
pointcut: (JoinPointDescriptor
targetClass: Engine targetSelector: #accelerateWheels:)
afterBlock: [:receiver :arguments :aspect |
"computation of the velocity according to the speed of the wheels"
velocity := ...]

PositionControlAB. PositionAndVelocity>> adviceComputeHeading

"AfterAdvice
pointcut: (JoinPointDescriptor
targetClass: Engine targetSelector: #setAnglewithHeading:)
afterBlock: [:receiver :arguments :aspect |
"computation of the heading according to the speed of the wheels"
heading := ...]

Figure 2: The velocity and the heading are computed by two advices adviceComputeVelocity and adviceComputeHeading, respectively.

aspect is defined. Outside this aspectbox, it is as if no aspect would have been applied to the base system.

The aspectbox SafetyControlAB defines the aspect PowerOff. This aspect has one advice, adviceDrivingControl that proceed a method call if the hasPower is true.

(Aspectbox named: #SafetyControlAB)
createAspectNamed: #PowerOff
instanceVariableNames: 'hasPower'.

SafetyControlAB.PowerOff>>
adviceDrivingControl
|
joinpoints | joinpoints := JointPointDescriptor
targetClasses: {Joystick . Engine . Wheels}. *AroundAdvice
pointcut: joinpoints
aroundBlock: [:receiver :arguments :aspect |
hasPower ifTrue: [ aspect proceed ]

Aspects PowerOff and PositionAndVelocity described above have a common pointcut: public method of the class Engine. Because these two aspects belongs to different aspectboxes (SafetyControlAB and PositionControlAB, respectively), they do not conflict with each other.

4.3 Executing Code in an Aspectbox

Triggering a program execution in an aspectbox is achieved by the method eval:.

(Aspectbox named: #SafetyControlAB) eval: [
| app |
app := SafetyApplication new.
app run].

The code above instantiates the class SafetyApplication and invokes the method run. The code invoked by this method run will benefit from aspects defined in SafetyControlAB (i.e., PowerOff). Similarly, an application invoked in the aspectbox PositionControlAB will benefit from PositionAndVelocity without being affected by SafetyControlAB.

4.4 Absolute Isolation of Aspects

It is widely accepted that encapsulating different functionalities of a system in distinct modular units aids their comprehensibility and maintainability (Parnas, 1972).

Figure 1 illustrates a modular architecture. Because it is closely linked to the physical and external physical mechanic events, the driving control unit needs special care and should not be altered by other units that are not necessary for its execution. Also, for safety reasons, the position control unit has to be built on top of the motion engine without affecting its execution. Different concerns composed into a system have to be well modularized and isolated from the base system.

The aspectboxes module system has the following properties:

- **Conflicts between aspects are avoided.** By living in different scopes, aspects are kept separated. Even if aspects defined in different aspectboxes have the same join points, there is no need to define precedence rules for composition ordering.

- **Minimal extension of the aspect language.** Combining the aspectboxes module system with AspectS (Hirschfeld, 2003) did not require any modification of the aspect language syntax. Static references contained in the definition of pointcuts are resolved using the classes visible in the aspectbox in which these pointcuts are defined in.
5 IMPLEMENTATION

A prototype of aspectboxes is implemented in Squeak. Figure 3 describes how the safety control and the position control are hooked into the driving control module.

AspectS. AspectS (Hirschfeld, 2003) is an approach to general-purpose aspect-oriented programming in the Squeak3 Smalltalk environment (Ingalls et al., 1997). It extends the Squeak metaobject protocol to accommodate the aspect modularity mechanism. In contrast to systems like AspectJ, weaving and unweaving in AspectS happens dynamically at runtime, on-demand, employing metaobject composition. Instead of introducing new language constructs, AspectS utilizes Squeak itself as its pointcut language. AspectS benefits from the expressiveness and uniformity of Squeak.

Activation blocks. AspectS uses Method Wrappers (Brant et al., 1998) to instrument both message sends and receptions. Such wrappers support execution of additional code before, after, around, or instead of an existing method. The core of the aspect activation mechanism is implemented in the isActive method of the class MethodWrapper. All additional code provided by a wrapper is to be activated only if all activation blocks associated with it evaluate to true. Activation blocks are treated as predicate methods, returning either true or false as the outcome of their execution.

Aspectboxes. The aspectboxes module system is fully integrated in the Squeak environment. When an aspect is woven, activation blocks are created and placed at join points shadows. When the control flow of the application reaches a join point, the isActive method is executed in order to determine if this potential join point is within the scope of an aspectbox defining this aspect to yield activation or not (i.e., if it is associated with the current control flow).

6 RELATED WORK

AspectJ. The pointcut language offered by AspectJ provides a mechanism to restrict a pointcut definition to a package or a class (i.e., within and withincode pointcut primitives). The purpose of these constructs is to restrict the location of join points between a base system and an aspect, however advices hooked at those join points remain globally visible. Therefore, the restricting pointcut primitives of AspectJ do not help in scoping an aspect application.

CaesarJ. Aspects, packages and classes are unified in CaesarJ (Aracic et al., 2006) under a single notion, a cclass. Aspect deployment can either be global or thread local. Aspectboxes promotes a syntactic scoping of aspects: an aspect is scoped to the aspectbox that defines it. In CaesarJ, an aspect is scoped to the thread it was installed in.

Classboxes. The Classbox module system allows a class to be extended by means of class member additions and redefinitions. These extensions are visible in a locally and well-delimited scope. Several versions of a same class can coexist at the same time in the same system. Each class version corresponds to a particular view of this class (Bergel et al., 2005).

Classboxes and aspectboxes have a common root which is the scoping mechanism for refinement. Whereas classboxes support structural refinement (i.e., class members addition and redefinition), aspectboxes offer a scoping mechanism for behavioral refinement.

Context-aware aspects. Context awareness promotes software program behaviour to depend on "context". Context-aware aspects (Tanter et al., 2006)
offers language constructs to handle contexts. A context is defined by the programmer as a plain standard object. The pointcut language is extended with primitives such as inContext(c) and createdInContext(c) that restrict a pointcut expression to a particular context c and to objects that were created in a context c, respectively.

Whereas context-aware aspects trigger the activation of aspects based on some arbitrary context activation function, aspectboxes promote the concurrent applications of aspects by restricting them to different scope.

Context-oriented programming. ContextL (Costanza and Hirschfeld, 2005), a CLOS-based implementation for Context-Oriented Programming, provides dedicated programming language constructs to associate partial class and method definitions with layers. Layers activation and deactivation is driven by the control flow of a running program. When a layer is activated, the partial definitions become part of the program until this layer is deactivated.

Whereas scoping software system refinement is the common problem for context-oriented programming and aspectboxes, the approaches are different. A layer in ContextL encapsulate structural definitions, whereas aspectboxes encapsulate behavioral definitions.

AWED. Aspects with Explicit Distribution (AWED) (Navarro et al., 2006) is an approach for defining crosscutting behaviour on remote locations (i.e., distributed applications). AWED is an aspect language supporting remote pointcuts, distributed advices and distributed aspects. A distributed aspect allows for state sharing and aspect instance to be distributed across multiple hosts.

7 CONCLUSION

Aspectboxes provide a new aspect modularity construct limiting the scope of aspect composition with a base software system. Modifications to the base system are visible only in the aspectbox the aspect is defined in. This allows one to deploy multiple concurrent modifications in the same base system, avoiding conflicting situations across aspectboxes.

In the work presented in this paper, an aspect cannot be imported from an aspectbox. The reason for this is that aspects are not generic (i.e., cannot be applied to other base systems). As future work, we plan to refine the notion of import to enable reuse of aspects within multiple aspectboxes.

Our prototypical implementation is based on AspectS. It integrates the composition mechanisms of AspectS and Classboxes to achieve the desired composition and scoping behavior.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support of the Science Foundation Ireland and Lero — the Irish Software Engineering Research Centre. We also like to thank Parinaz Davari and Daniel Rostrup for their valuable comments.

REFERENCES


ON ABILITY OF ORTHOGONAL GENETIC ALGORITHMS FOR THE MIXED CHINESE POSTMAN PROBLEM

Hiroshi Masuyama  
Information and knowledge Engineering, Tottori University, Koyama-cho Minami 4-101, Tottori, Japan  
masuyama@ike.tottori-u.ac.jp

Tetsuo Ichimori  
Information Systems, Osaka Institute of Technology, Kitayama 1-79-1, Hirakata, Japan

Toshihiko Sasama  
Information and knowledge Engineering, Tottori University, Koyama-cho Minami 4-101, Tottori, Japan

Keywords: Orthogonal design, Genetic algorithm, Chinese postman problem, Postman’s route.

Abstract: The well known Chinese Postman Problem has many applications, and this problem has been proved to be NP-hard in graphs where directed and non-directed edges are mixed. In this paper, in order to investigate the salient feature of orthogonal design, we designed a genetic algorithm adopting an orthogonal crossover operation to solve this (mixed Chinese Postman) problem and evaluate the salient ability. The results indicate that for problems of small sizes, the orthogonal genetic algorithm can find near-optimal solutions within a moderate number of generations. We confirmed that the orthogonal design shows better performance, even for graph scales where simple genetic algorithms almost never find the solution. However, only the introduction of orthogonal design is not yet effective for the Chinese Postman Problem of practical size where a solution can be obtained in less than 104 generations. This paper concludes that the optimal design scale of orthogonal array to this mixed Chinese Postman Problem does not conform to the same scale as the multimedia multicast routing problem.

1 INTRODUCTION

The Chinese Postman Problem, as is well known, is to find the shortest route in a graph that uses every arc (directed or non-directed edge) and gets back to where it started. For example in the non Eulerian graph shown in Fig.1, since Postman’s route traverses every arc at least once, the Postman must pass doubly through an arc of weight 6. By duplicating some arcs, the non Eulerian graph can have at least one Postman’s route. In general, if a given graph is a non Eulerian graph, it can be said that the optimum solution of the Chinese Postman Problem is a route where the total weight of duplicated arcs is the minimum. When a given graph is an Eulerian graph, the solution is uniquely determined.

Though this problem has many applications, including robot exploration and analyzing interactive systems and web site usability (Thimbleby, 2003), this problem has been proved to be NP-hard. The multimedia multicast routing problem is also NP-hard. Paper (Zhang and Leung, 1999) proposed an orthogonal genetic algorithm for this latter problem, and concluded on the basis of solving a benchmark test problem, that for practical problem sizes the orthogonal genetic algorithm can find near-optimal solutions within a moderate number of generations. Its salient feature is to incorporate an experimental design method called orthogonal design into the crossover operation. In order to further investigate this salient feature of orthogonal design, which was applied to the sampling of genes from the parents for crossover, we will design a genetic algorithm adopting an orthogonal crossover operation to solve the mixed Chinese Postman Problem and evaluate the salient ability.

For the problem which we treat is called the Chinese postman problem on mixed networks (WCPP), heuristic solution procedures have been proposed to solve approximately (Edmond and Johnson, 1979)(Pearn and Liu, 1995)(Frederickson, 1979).
1. [Start] Generate random population of $t$ chromosomes (suitable solutions for the problem).
2. [Fitness] Evaluate the fitness $f(x)$ of each chromosome $x$ in the population.
3. [New population] Create a new population by repeating following steps until the new population is complete.
   1. [Selection] Select two parent chromosomes from a population according to their fitness (the better their fitness, the better their chance of being selected).
   2. [Crossover] Use crossover probability to cross over the parents and form a new offspring (children). If no crossover was performed, the offspring would be an exact copy of the parents.
   3. [Mutation] Use mutation probability to mutate new offspring at each locus (position in the chromosome).
   4. [Accepting] Place new offspring in a new population.
4. [Replace] Use newly generated population to continue the algorithm.
5. [Test] If the end condition is satisfied, stop, and return the best solution in the current population.

Figure 2: A basic process of genetic algorithm.

Figure 1: An example of route in Chinese postman problem.

2 GENETIC ALGORITHM AND ORTHOGONAL ARRAY REPRESENTATION

A genetic algorithm (GA) is a heuristic approach used to find approximate solutions to knotty problems through application of the principles of biological evolution. Genetic algorithms make the best of biologically derived approaches such as inheritance, mutation, natural selection, and recombination (or crossover). Genetic algorithms are a particular class of evolutionary algorithms where a population of abstract representations (called chromosomes) of candidate solutions (called individuals) evolve into better solutions. That is, information treated in GA can be classified into two structures: phenotype and genotype. Phenotype represents information in the biological world, and genotype represents information in a population of chromosomes. By encoding, the information in a phenotype can be transferred into a genotype, and by decoding the opposite occurs. We will chiefly consider that in a genotype. Even though some different encodings are possible, in general the solutions are represented in binary strings of 0s and 1s.

In this mixed Chinese postman problem, we will treat a given graph $G$ whose every arc is a directed edge or a non-directed edge. We will consider a directed graph $G'$ where every non-directed edge in the given graph is changed into two directed edges with different directions each other. Then, we can make our chromosome type as an integer string whose element means the number of times that the postman passes the arc. The length of a string is the number of edges of $G'$. Therefore, in this mixed Chinese postman problem, the solutions are represented in strings of integer. The evolution starts from a population of completely random individuals and goes on in generations. In each generation, the fitness of the whole population is evaluated, and multiple individuals are stochastically selected from the current population (by judging their fitness) and modified (so called by mutation or recombination) to form a new population, which becomes current in the next iteration of the algorithm. The general process of GA is known, and is shown in Fig.2.

Orthogonal array was developed to find the smallest, yet most cost effective, and therefore best, combination by which many and consumptive
combinations can be avoided (Fang and Wang, 1994). An orthogonal array is represented in Table 1-1 by L₉(3⁴), where 3, 4, and 9 mean the number of kinds of entries, columns, and rows, respectively. In general, we let Lₘ(ₙᵏ) denote an orthogonal array for k factors, n levels, and m combinations of level to be tested. Orthogonal arrays can be systematically built. Label L comes originally from a “Latin” square, which is defined as a matrix where no two entries in a row (or a column) have the same value. It has been proved that the orthogonal design is optimal for use as an additive model and a quadratic model, and that the selected combinations are good representatives for all the possible combinations (Wu, 1978). The problem of building an orthogonal array is the same as the problem of finding m nodes which are at the maximum distance between any pair in the (k log n)-dimensional hypercube. Table 1-2 shows 9 nodes corresponding to 9 combinations (on an 8-dimensional hypercube) in Table 1-1.

### Table 1-1: A representation of orthogonal array L₉(3⁴).

<table>
<thead>
<tr>
<th>Combination</th>
<th>Factor1</th>
<th>Factor2</th>
<th>Factor3</th>
<th>Factor4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2nd</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>3rd</td>
<td>X</td>
<td>Z</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>4th</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
<td>Z</td>
</tr>
<tr>
<td>5th</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>6th</td>
<td>Y</td>
<td>Z</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>7th</td>
<td>Z</td>
<td>X</td>
<td>Z</td>
<td>Y</td>
</tr>
<tr>
<td>8th</td>
<td>Z</td>
<td>Y</td>
<td>X</td>
<td>Z</td>
</tr>
<tr>
<td>9th</td>
<td>Z</td>
<td>Z</td>
<td>Y</td>
<td>X</td>
</tr>
</tbody>
</table>

### Table 1-2: 9 nodes corresponding to 9 combinations on a (k log n)-dimensional hypercube.

<table>
<thead>
<tr>
<th>Combination</th>
<th>Factor1</th>
<th>Factor2</th>
<th>Factor3</th>
<th>Factor4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2nd</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3rd</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4th</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5th</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6th</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7th</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8th</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9th</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### 3 EXPERIMENTAL DESIGN METHODS

In this section, we introduce the concept of experimental design methods for our experiment mentioned later.

#### 3.1 Phenotype

Roads are modeled as a graph G = (V,E), where V is a set of nodes and E is a set of arcs. By making every non-directed arc express as two directed arcs, change a given graph G into a directed graph G’. As mentioned before, the optimum solution is a route where the total weight of duplicated arcs is the minimum. We number the edges in G’ from 1 to t. We can then represent any route R of the graph G’ as an t-tuple (e₁, e₂, ..., eₜ), where any element (which means a gene) eᵢ is defined as follows:

\[ eᵢ = \text{The number of times that the postman passes arc } i \]

From the above definition, eᵢ is a non negative integer and can become 0 if eᵢ is represented for one of two directed edges changed from non-directed edge in G. For example, assuming that edge with weight 4 is r₄, R in Fig.1(b) can be represented as (eᵣ₄, eᵣ₈, eᵣ₅, eᵣ₉, eᵣ₃, eᵣ₃, eᵣ₂, eᵣ₆) = (1, 1, 1, 1, 1, 0, 1, 2). Since R in Fig.1(b) is a Postman’s route, then this phenotype (1, 1, 1, 1, 1, 0, 1, 2) is a solution.

#### 3.2 Fitness

The Postman can return to the starting point if G is an Eulerian graph. If G is a non Eulerian graph, in order to return to the starting point the postman must traverse some arcs more than once. In general, if G is a non Eulerian graph, it can be said that the optimum solution of the mixed Chinese Postman Problem is a route where the total weight of duplicated arcs is the minimum. When G is an Eulerian graph, the solution is uniquely determined.

In order to set the fitness of the optimum solution to maximum, we will prepare the following function f of fitness:

\[ f = \begin{cases} 1 / (\text{total weight of duplicating arcs}) & \text{if } G \text{ has Postman's route} \\ 0 & \text{otherwise} \end{cases} \]

#### 3.3 Orthogonal Crossover

In order to fit the mixed Chinese postman problem, we interpret orthogonal array Lₘ(ₙᵏ) to be an orthogonal array for k factors divided from n levels (parent chromosomes), and m combinations of levels (samplings at the time of crossover). Let orthogonal
array $L_4(2^l)$ be shown in Table 2. Obeying the above interpretation of $L_4(2^l)$, 2 parent chromosomes are divided into 3 factors each, and we obtain 4 new chromosomes as shown in Fig.3. In the case of $L_4(3^3)$, we obtain 9 new chromosomes from 4 parent chromosomes as shown in Fig.4.

<table>
<thead>
<tr>
<th>Combination</th>
<th>Factor1</th>
<th>Factor2</th>
<th>Factor3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2nd</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>3rd</td>
<td>Y</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>4th</td>
<td>Y</td>
<td>Y</td>
<td>X</td>
</tr>
</tbody>
</table>

![Figure 3: Orthogonal crossover for $L_4(2^l)$](image)

Each $X$ and $Y$ are corresponded to a random number in the same range of the random numbers. Randomly correspond to a random number in the same levels in orthogonal array select, let each one of the random numbers is divided into the number of levels of orthogonal array. In addition, for each one of the same levels in orthogonal array select, let randomly correspond to a random number in the same range of the random numbers.

Let us take $L_4(2^l)$ shown in table 3 and $t=12$ as an example. Since a $t$-tuple is divided into 3 factors, then we can represent 4 chromosomes based on $L_4(2^l)$ as follows:

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>X</td>
<td>X</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>2nd</td>
<td>Y</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3rd</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>4th</td>
<td>Y</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Each $X$ and $Y$ are corresponded to a random number in the ranges 0–6 and 7–12, respectively. Then, we obtain the following initial population of chromosomes (4 chromosomes) as an example.

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>2nd</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>7</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>3rd</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>11</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>4th</td>
<td>7</td>
<td>11</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

### 4 EXPERIMENT

In this section, we will explain the experiments we performed.

#### 4.1 Graphs

We used random graphs shown in Table 4. We applied the Genetic algorithm 30 times to each graph and evaluated the mean values.

| $|V|$ | $|E|$ | Total weight |
|-----|-----|-------------|
| 5   | 7   | 44          |

### 4.2 Orthogonal Genetic Algorithm (OGA)

We developed the following algorithm, as shown in Fig.5, where an initial group starts from a population of completely randomly generated individuals, and...
the probabilities of crossover and mutation are 1.0 and 0.5, respectively.

4.3 Simple Genetic Algorithm (SGA)

The simple genetic algorithm adopts a one-point crossover in Step 2.1 instead of orthogonal crossover.

4.4 Another Orthogonal Genetic Algorithm using Orthogonal Array in the Creation of the Initial Population of Chromosomes (OGA²)

This algorithm uses an orthogonal array also in Step 1 in Fig.5 instead of random number, and the utilization method is shown in 3.4.

4.5 Results

Fig.6-1 shows the relationship between the number of obtained Eulerian graphs and the number of generations. On the other hand, Fig.7-1 shows the relationship in the case where orthogonal array is used in the creation of initial population of chromosomes. Fig.6-2 shows the relationship between the obtained minimum weight of the Postman route and the number of generations. Fig.7-2 shows the relationship in the case where orthogonal array is used in the creation of initial population of chromosomes. These results mean that our orthogonal genetic algorithm shows better performance, especially in L9(3⁴). SGA almost never finds the solutions for Problem 3 where the number of edges is 7.

For reader’s information, we show the relationship between the number of generations and the computation time required in 3 algorithms (SGA, OGA, and OGA²) in Tables 5 and 6.

In this mixed Chinese postman problem we treated, in less than 10⁶ generations we can obtain a solution in graphs with nodes of less than 11. However, we can’t obtain a solution in 2 or 3 days for the larger sizes.

For reference we will show the data in the case of non directed graphs G''=(V'', E''), where |V''|=20, |E''|=30, total weight=178. The Chinese Postman problem for non directed graphs belongs in Class P. Figs.8-1 and 8-2 show the relationship between two numbers of obtained Eulerian graphs and generations, and the relationship between the obtained minimum weight of the Postman route and the number of generations, respectively.

| Inputs: | Graph G=(V,E), edge cost C(e) and node degree deg(v) |
| Output: | binary strings representing the Euler route |
| Step1) | **Initialization** |
| | Randomly create an initial generation of N binary string P₀ = { X₁, X₂, ..., X_N }, X = { x₁, x₂, ..., x_k } and initialize the generation number gen to 0. |
| | *X is a chromosome, k = |V| |
| Step 2) | **Population Evolution** |
| | WHILE ( gen < MAX_GEN ) |
| | BEGIN |
| | DO N/2 times |
| | BEGIN |
| | **Orthogonal Crossover** |
| | Randomly select n parents strings from P_gen and perform orthogonal crossover on them to generate m offspring o₁, o₂, ..., o_m. |
| | END |
| | **Mutation** |
| | To perform mutation of offspring, flip every bit in this string with a small probability p. |
| | END |
| Step 3) | Increment the generation number gen by 1. |

Figure 5: A orthogonal genetic algorithm.

5 CONCLUSION

In order to investigate the salient feature of orthogonal design, we designed a genetic algorithm adopting an orthogonal crossover operation in the mixed Chinese Postman Problem and evaluated the salient ability. The orthogonal design shows better performance, even for graph scales where simple genetic algorithms almost never find the solution. The experimental results show that, for problems of non practical sizes, the orthogonal genetic algorithm using the orthogonal array L₉(3⁴) can find close-to-optimal solutions within a moderate number of generations. This optimal scale of orthogonal array was confirmed for the multimedia multicast routing problem of practical size (Zhang and Leung, 1999). However, this orthogonal design is not yet effective for the mixed Chinese Postman Problem of practical sizes. For more effective computation, our one possible extension of this research can be considered as to incorporate the orthogonal array into the
Table 5: Computation time required at 1000–10000 generations.

<table>
<thead>
<tr>
<th>generation</th>
<th>SGA</th>
<th>OGA using L_d(2^3)</th>
<th>OGA using L_d(3^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>247.1</td>
<td>279.8</td>
<td>437</td>
</tr>
<tr>
<td>2000</td>
<td>252.5</td>
<td>286.4</td>
<td>447</td>
</tr>
<tr>
<td>3000</td>
<td>257.8</td>
<td>292.6</td>
<td>456.6</td>
</tr>
<tr>
<td>4000</td>
<td>263.3</td>
<td>298.4</td>
<td>465.8</td>
</tr>
<tr>
<td>5000</td>
<td>268.3</td>
<td>304.4</td>
<td>475</td>
</tr>
<tr>
<td>6000</td>
<td>273.8</td>
<td>310.3</td>
<td>484.4</td>
</tr>
<tr>
<td>7000</td>
<td>279.1</td>
<td>316.2</td>
<td>493.6</td>
</tr>
<tr>
<td>8000</td>
<td>284.5</td>
<td>322.1</td>
<td>502.9</td>
</tr>
<tr>
<td>9000</td>
<td>289.9</td>
<td>328.1</td>
<td>512.5</td>
</tr>
<tr>
<td>10000</td>
<td>295.3</td>
<td>333.9</td>
<td>521.7</td>
</tr>
</tbody>
</table>

Table 6: Computation time required at 1000–10000 generations.

<table>
<thead>
<tr>
<th>generation</th>
<th>OGA using L_d(3^3)</th>
<th>OGA^2 using L_d(2^3)</th>
<th>OGA^2 using L_d(3^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (sec)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>437</td>
<td>428.3</td>
<td>433.4</td>
</tr>
<tr>
<td>2000</td>
<td>447</td>
<td>437.7</td>
<td>443.2</td>
</tr>
<tr>
<td>3000</td>
<td>456.6</td>
<td>447.3</td>
<td>452.7</td>
</tr>
<tr>
<td>4000</td>
<td>465.8</td>
<td>456.4</td>
<td>462.3</td>
</tr>
<tr>
<td>5000</td>
<td>475</td>
<td>465.7</td>
<td>471.6</td>
</tr>
<tr>
<td>6000</td>
<td>484.4</td>
<td>474.7</td>
<td>480.9</td>
</tr>
<tr>
<td>7000</td>
<td>493.6</td>
<td>483.9</td>
<td>490.3</td>
</tr>
<tr>
<td>8000</td>
<td>502.9</td>
<td>493.1</td>
<td>499.5</td>
</tr>
<tr>
<td>9000</td>
<td>512.5</td>
<td>502.2</td>
<td>508.6</td>
</tr>
<tr>
<td>10000</td>
<td>521.7</td>
<td>511.4</td>
<td>517.9</td>
</tr>
</tbody>
</table>
experimental design methods of setting an initial
group of populations. We performed this extension.
The experiment results show no innovative improvement.

REFERENCES

Thimbleby, H., 2003. The directed Chinese postman
problem. In Software - Practice & Experience, Vol.33,
No.11, pp.1081-1096.

algorithm for multimedia multicast routing. In IEEE
Trans. on Evolutionary Computation, Vol.3, No.1,
pp.53-62, April.

Fang, K.T. and Wang, Y., 1994, Number-Theoretic

experimental design. In Acta Mathematicae Sinica,
Vol.1, No.4, pp.283-299.

Sasama, T., Masuyama, H. and Ichimori, T., 2002. On
Fault Tolerance of Hypercubes using Subcubes. In Int.
Journal of Reliability, Quality and Safety Engineering,
Vol.9, No.2, pp.151-161.

for the Multiple Destination Routing Problem. In
IEEE Trans. on Evolutionary Computation, Vol.2,
No.4.

Algorithm for Multimedia Multicast Routing. In IEEE

Edmond, J. and Johnson, E., 1979. Matching, Euler tours,
and the Chinese postman. In Mathematical
Programming, No.5, pp.538-554.

Chinese postman problem on mixed networks. In
Computers & Operations Research, Vol.22, No.5,
pp.479-489, May.

Frederickson, G.N., 1979. Approximation algorithms for
Vol.26, pp.538-554.

Figure 8-1: Relationship between two numbers of obtained
Eulerian graphs and generations.

Figure 8-2: Relationship between the obtained minimum
weight of the Postman route and the number of
generations.
Keywords: JAAS, RBAC, authorization, Java, AspectJ, AOP, Zs.

Abstract: This paper proposes Zás, a novel, flexible, and expressive authorization mechanism for Java. Zás has been inspired by Rammivas Laddad’s proposal to modularize Java Authentication and Authorization Services (JAAS) using an Aspect-Oriented Programming (AOP) approach. Zás’ aims are to be simultaneously very expressive, reusable, and easy to use and configure. Zás allows authorization services to be non-invasively added to existing code. It also cohabits with a wide range of authentication mechanisms.

Zás uses Java 5 annotations to specify permission requirements to access controlled resources. These requirements may be changed directly during execution. They may also be calculated by client supplied permission classes before each access to the corresponding resource. These features, together with several mechanisms for permission propagation, expression of trust relationships, depth of access control, etc., make Zás, we believe, an interesting starting point for further research on the use of AOP for authorization.

1 INTRODUCTION

This paper proposes a novel, flexible, and expressive authorization mechanism. Its advantages stem mainly from the fact that an AOP approach is used, allowing it to address some of the problems found in industry standards like JAAS (Lai et al., 1999; Coté, 2006; Oaks, 2005). AspectJ (AspectJ Team, 2006) has been used to develop the proposal.

AOP is strong in terms of reduction of code scattering and tangling, provides for the separation of crosscutting concerns from the core code, and nicely integrates with the expressiveness of Java 5 annotations. This, together with our practical interest in authorization services for Java applications, led us to attempt to develop a new, aspect-oriented authorization mechanism, called Zás. Our aims were to make Zás simultaneously very expressive, allowing programmers to state very clearly what they mean, and independent from the business context, though being possible to make it business aware, allowing the separation of particular permission specifications from the authorization concerns embedded into the code by programmers. Another of our goals was to make the application of Zás to already existing code as simple as possible and, if necessary, totally non-invasive.

Expressiveness requires code which is as clear and simple as possible. We do not want to tangle business code with code related to the checking of user permissions. We also do not want permission checking code scattered all over the application, in every method requiring verification of authorized access. In other words, we want to modularize, into aspects, the crosscutting concerns related to authorization. On the other hand, the programmer should be able, though not required, to guide the application of the authorization concerns to his own code. In this sense, one might say that one of the tenets of AOP, viz. obliviousness (Filman and Friedman, 2005), is violated. However, annotations may be though of as allowing the programmer to express in the code its required semantics, still oblivious of the exact way in which this semantics will actually be implemented.

According to (Clifton and Leavens, 2002), aspects can be divided into two categories: assistants and spectators. They suggest that assistance should be explicitly accepted by a module. Once a module accepts assistance of an aspect, then the aspect is allowed to advise that module. Annotations can be seen as a way to express assistance acceptance. Authorization annotations, as proposed by Zás, will thus acknowledge specific semantics for, say, a method, and implicitly
accept (or rather, require) assistance of a corresponding aspect implementing that semantics.

In (Reccebli, 2005), Recebli proposes a different classification related to the role of aspects in software systems from a higher-level engineering point of view. He proposes the division into integral and attachable aspects. According to him, we can say that the aspects within our approach are attachable, since they can be removed from an application, without in any way changing the correctness of its business implementation (except, of course, in what concerns authorization).

The JAAS authentication service, based on PAM\(^1\) (Samar and Lai, 1996), provides an abstraction layer that greatly simplifies changes in the actual authentication method used. However, since this work was motivated by the need to add authorization to the Heliopolas Web application,\(^2\) which already possessed its own authentication code, our main focus was solely on authorization concerns. Nevertheless, the developed solution can be seamlessly integrated with a wide range of authentication mechanisms.

This work was inspired by Laddad’s proposal (Laddad, 2003) to use AOP to modularize JAAS-based authentication and authorization. The main interest of his proposal, at least from our point of view, is related to the authorization concerns. Our work is thus based in Laddad’s, extending further the modularization of authorization concerns, thus reducing code tangling and scattering, and reducing the configuration effort required from the programmers.

This paper is structured as follows. The next section will review some of the existing Java-based authorization mechanisms. It is followed by a detailed description of the requirements that have been used as guidelines to develop Zás. And finally, conclusions will be drawn and some possible directions for further work will be pointed to.

The Zás source code and related projects can be downloaded from https://svn.ci.iscte.pt/zenida, namely the Zás source code and the Web application used as a case study for this research. Further details about Zás, its implementation and the case study results can be found in (Zenida et al., 2006).

2 AUTHORIZATION SOLUTIONS IN JAVA

Authorization is not a new research topic. There are many different proposals and tools readily available, ranging from ad hoc solutions, where the developer

\(^1\)Pluggable Authentication Modules

\(^2\)See http://heliopolis.iscte.pt (the source code uses Zás and is available in https://svn.ci.iscte.pt/Heliopolis/trunk/).

implements everything from scratch, to complete solutions like JAAS, and from OOP approaches to approaches where the power of AOP is leveraged.

Our main interest while studying existing authentication and authorization mechanisms was JAAS, since it is a standard for authentication and authorization services in Java (Sun Microsystems, Inc., 2006) and an integral part of the JDK.

JAAS is not as flexible as we would like it to be. Its use requires considerable configuration effort (Oaks, 2005). For example, security policy files have to be used in order to specify the principals and what they are permitted to do. For example:

grant Principal sample.principal.Principal "user" {
    permission test.Permission "perm";
};

Besides, and importantly, the permissions can not be changed at runtime. This is a serious restriction for dynamic applications, where an administrator must be able to add users and their corresponding permissions during the operation of the system. It is possible to use a database for this task, as in the example provided in (Coté, 2006), increasing the flexibility of the system by allowing the privileges of the principals to be specified at runtime. However, such a solution requires the use of a specific database model that, for already existing systems, may not be easy to accomplish.

The original JAAS model is implemented with an OO approach, thus being prone to the common problems of code scattering and tangling: code must be added to the business classes in order to implement authorization.

```java
public class MyClass {
    public void businessMethod() {
        AccessController.checkPermission(
            new MyPermission("aPermission"));
        // business code
    }

    public static void main(String args[]) {
        // authentication code
        MyClass a = new MyClass();
        Subject authenticatedSubject = lc.getSubject();
        Subject.doAsPrivileged(
            authenticatedSubject,
            new PrivilegedAction() {
                public Object run() {
                    a.businessMethod();
                }
            }, null);
    }
}
```

Clearly, the authorization code is entangled with business code, both in the code requesting access to
the resource (the caller code) and in the resource code itself (the callee code). Moreover, authorization code will be scattered through the application, since it must be used wherever access control is required. Both problems can be fixed using AOP, as shown in (Laddad, 2003). Laddad proposes an aspect-oriented approach to the application of JAAS that significantly simplifies the code required for access control, though only at the caller:

```java
public class MyClass {
    // as before
    public static void main(String args[]) {
        MyClass a = new MyClass();
        a.businessMethod();
    }
}
```

We still need to call the `checkPermission()` method in the business methods. This can be avoided if we use the expressiveness of Java 5 annotations and modularize that call into an aspect responsible for authorization verification:

```java
@AccessControlled(
    requires = "aPermission",
    permissionClass = MyPermission.class
)
public void businessMethod() {
    // business code
}
```

The use of annotations clearly improves the quality of the code, augmenting its expressiveness while reducing scattering and entanglement. However, by itself it does not decrease the required configuration effort nor makes access control dynamic. Zás, as will be seen in the next sections, does.

3 REQUIREMENTS

Originally, the implementation of authorization in Heliópolis was a clear case of a simplistic implementation of Yoder’s “Limited View” pattern (Yoder and Barcalow, 1997), where appropriate menu options were hidden from non-authorized users. Our aim was thus to solve the authorization problem not by merely limiting the view of each user to whatever she is allowed to view or manipulate, but mainly by making sure, at the business layer itself, that no user can ever gain unpermitted access to any resource that is outside the privileges associated with his roles within the system.

For the reasons stated in the previous section, we were not satisfied with the available solutions to this problem. Zás, as described in this paper, was thus developed as a non-ad hoc solution to the authorization problem fulfilling the following broad requirements:

1. It should be independent and compatible with the simultaneous use of JAAS, especially with its authentication services.
2. Its authorization services should require no more from the application model than access to the current principal’s permissions. It should thus support the RBAC (Ferraiolo et al., 2006; Sandhu et al., 1996) model, though never dealing directly with roles itself.
3. It should greatly simplify the code of client applications, as compared with alternative solutions.
4. It should be as non-invasive as possible, allowing business code to concentrate on the business logic, allowing the programmer to specify access requirements within the code, if she so desires, but also to completely separate access control from the business logic code.
5. It should require less configuration effort than the alternatives.
6. It should allow dynamic changes to the resources’ access requirements.

Since Zás was meant to be a Java/AspectJ library of classes and aspects for use in Java applications, the requirements above were further refined into the following detailed requirements:

1. The access requirements for each such resource should be specifiable using Java 5 annotations.
2. The resources whose access should be controlled are represented by constructors, methods, and attributes.
3. It should be possible to force the propagation of the access requirements of a resource to all its members. For instance, from a package to all its types and nested packages, and from a class to all its (non-private) methods and attributes.
4. It should not be possible to propagate access control specification to resources explicitly marked as having no access control.
5. It should be possible to define access requirements either next to the corresponding resource definition (invasive usage), centralized in a single or in several access requirement definition aspects (non-invasive), or both.
6. It should be possible to use outside sources of permission requirements, such as property files. Permission requirements should be possible to change dynamically.
7. It should allow the definition of access requirements using boolean expressions involving permission names.
8. It should allow the quantification of the definition of access requirements using wildcards.
9. It should allow the specification of the required depth of access control as either deep or shallow.
10. When shallow access control is required, it should be possible to specify the degree of suspiciousness of a resource.
11. Special cases should be provided to bypass access control, viz. using privileged methods and trusted classes.
12. It should be easy to add authorization features to existing projects.

The next sections go through several of these requirements, exemplifying their impact in the client code, and thus clarifying the importance of the requirements themselves. Notice, however, that at the current state not all requirements have been implemented and some are only partially implemented. The status of development will be stated wherever appropriate.

### 3.1 Annotations (1 and 2)

Zás should allow the programmer to guide the application of aspects through the annotation of the non-private\(^3\) resources where access control is required:

```java
import pt.iscte.ci.zas.authorization.*;
public class MyClass {
    @AccessControlled(
        requires = "aPermission"
    )
    public void foo() {} } } } }
```

The previous code explicitly states that access to method `foo()` is restricted to principals having permission `aPermission`. When not specified in the annotation, the access requirements correspond to a single permission whose name is the signature of the method without the return type.\(^4\) Hence, the permission required to call `foo()` as defined in `mypackage.MyClass.foo()`.

It should also be possible to annotate attributes, similarly to what happens for methods.\(^5\)

Access requirements should always be filtered by method `getRequirements()` of the `Permission` class:

```java
package pt.iscte.ci.zas.authorization;
public class Permission {
    public String getRequirements(
        String currentRequirements,
        JoinPoint joinPoint,
        JoinPoint.StaticPart
        enclosingStaticPart) { return currentRequirements; }
}
```

Before each access to a protected resource, this method shall be passed the current access requirements, which the default implementation will simply return, as well as the execution context of the access, including the caller and callee objects.

It should be possible to provide access control specifications with client classes extending `Permission` and overriding `getRequirements()`:

```java
package mypackage;
public class MyClass {
    @AccessControlled(
        permissionClass = MyPermission.class
    )
    protected void foo() {} } } }
```

Hence, arbitrary client code may be executed during access control, making it possible to add business specific access control methods to Zás.

### 3.2 Propagation (3 and 4)

Zás should provide a mechanism allowing access control specifications to be propagated to members of the corresponding resource, if any. For instance, in

```java
package mypackage;
public class MyClass {
    @AccessControlled(
        requires = "aPermission",
        depth = Depth.SHALLOW
    )
    public class MyClass {
        public void foo() {} } } }
```

Hence, arbitrary client code may be executed during access control, making it possible to add business specific access control methods to Zás.

\(^3\)Private “resources” are implementation details.
\(^4\)Using complete signatures as permission names guarantees that overloaded resources are distinguished.
\(^5\)The current version of Zás does not distinguish between sets and gets, as it should.
controlled. The second one is dynamic, and leads to all non-private members of an access controlled resource that have not been explicitly marked as being either access controlled or not access controlled to dynamically inherit the required permissions from the enclosing resource (see Section 3.4).

The current version of Zás still does not provide the same mechanisms in the case of attributes. This problem will be solved in the near future.

Also, since the current version of AspectJ (AspectJ Team, 2006) does not allow the capture of package annotations, Zás still does not provide the inheritance mechanism for packages from the source code.\(^6\)

### 3.3 Location (5)

Usually resources requiring access control are directly annotated as such, i.e., their definition is directly annotated. This requires source code invasion and leads to scattering the meta-information related to access control concerns, which in some cases may be considered a bad practice.\(^7\)

It is possible to use AspectJ ITDs\(^8\) to inject annotations in types, methods, attributes, etc. Hence, it is possible to modularize all access control specifications in a single aspect, solving the problem of scattering meta-information.

Just as Java prohibits double annotations, AspectJ prohibits the injection of an annotation already present in the source code, next to the resource definition. Hence, the two approaches may be used together without any problem: the compiler will issue an error in case of a collision.

### 3.4 Dynamic Permissions (6)

Access control specifications should specify initial permissions, changeable at runtime. That is, permission requirements should be dynamic, while the access control character of resources should be static.

In conjunction with the ability to use wildcards (see Section 3.5) both to specify permissions and to specify the resources to which the permissions apply, this requirement makes it possible to dynamically load permissions specification from a generic input stream (connected to, e.g., access control property files), thus allowing permissions to be changed dynamically and easily by a system operator. For instance,

```java
public void foo(String s) {}
``` specifies that \(\text{foo}\) is access controlled and initially requires permission \(\text{mypackage.MyClass.}\text{foo}(\text{String})\). It should be possible to change the required permission using a properties file:

```java
mypackage.MyClass.foo(String) = \text{foo}
```

In this case, after loading the properties file, the required permission for calling \(\text{foo()}\) is no longer \(\text{mypackage.MyClass.foo(String)}\), but \(\text{foo}\). Of course, the same effect should be obtained by directly calling a permission changing method of Zás:

```java
import pt.iscte.ci.zas.authorization;
...
AccessController.addAccessControl(
    "mypackage.MyClass.foo(String)",
    "\text{foo}\"
);
```

The use of external sources of permission requirements allows them to be provided at the appropriate granularity level. For example,

```java
mypackage.MyClass.foo() = \text{foo} \text{ bar}
mypackage.MyClass.* = \text{bar}
mypackage.*() = \text{foo}
```

which might be found in an access control property file, states that all access controlled methods without any parameters within package \(\text{mypackage}\) will require permission \(\text{foo}\), with the exception of those within class \(\text{MyClass}\), which require permission \(\text{bar}\). Again, method \(\text{MyClass.foo()}\) is an exception, since it requires either permission \(\text{foo}\) or permission \(\text{bar}\) (see Section 3.5). The order is relevant because \(\text{Zás}\) will always look for the first occurrence of a matching signature and load the corresponding permission specification.

### 3.5 Expressions (7 and 8)

It should be possible to compose Boolean permission expressions, both in-code as initial permission requirements, and dynamically (e.g., inside property files). For instance, in the access control specification

```java
@AccessControlled(
    requires =
        "\text{aPermission} \text{ or} !\text{anotherPermission}\"
)
public void foo() {}
```

the permission expression requires any principal calling \(\text{foo()}\) either to have permission \(\text{aPermission}\) or to lack permission \(\text{anotherPermission}\).

Currently, \(\text{Zás}\) supports operators \(\text{ and} \) ("or"), \(\text{ ||} \) ("and"), and \(\text{ !} \) ("not"), as well as the use of parentheses to control evaluation order.

Regular expressions should also be possible in permissions expressions. For example, using
@AccessControlled(requires = "perm\*")
public void foo() {}

any call to foo() would require a principal having at least one permission whose name starts with perm (e.g., perm or permission). Notice that regular expressions introduce a form of quantification into Zás. In this case they introduce existential quantifiers into permission requirements.

Wildcards should also be possible when dynamically specifying permission requirements, of course. In this case, however, they can also be used to specify multiple resources in a single step, as shown in the last example of Section 3.4. This introduces the notion of universal quantifiers into Zás.

### 3.6 Depth (9 and 10)

By default, access control should be applied for all accesses to access controlled resources, regardless of the context. Regardless, namely, of the controls which have already been performed in upper levels of the current call stack. This is usually the safest option and thus the most desirable default. However, occasionally it may be necessary to turn off access control in the control flow of a given method execution.

The @AccessControlled annotation’s element depth represents the level of access control. In a way reminiscent of copy depth, access control is applied to method execution either in a Depth.SHALLOW or in a Depth.DEEP manner. Shallow access control means that if access to a method is granted to a principal, it will also be granted to its complete flow of control, effectively turning off access control during its execution. On the contrary, if access to a method specifying deep access control, which is the safe default, is granted to a principal, it will not be automatically granted to all other accesses to resources in the method’s control flow.

For example, in

```java
public class A {
    @AccessControlled
    public void foo() {
        new B().bar();
    }
}
public class B {
    @AccessControlled
    public void bar() {
        new C().baz();
    }
}
public class C {
    @AccessControlled(suspicious = true)
    public void baz() {}
}
```

a call to foo() will fail if the principal does not have permission C.baz(): it is not sufficient for him to have permission A.foo(), since baz() is suspicious. On the other hand, permission B.bar() is not necessary when the call is performed in the flow of control of foo(), since bar() is unsuspecting and foo()'s access control is shallow.

### 3.7 Bypasses (11)

Zás should provide two methods to bypass access control. The first is more dangerous, and should be used with care: it should be possible to annotate some methods as privileged, i.e., as turning off access control to calls within their control flow. The difference between calling a privileged method and calling a method with shallow access control is that the first call always succeeds, while the success of the second one depends (solely) on the current principal having permission to make the call.

The second required bypassing mechanism, trust, is more disciplined and less dangerous. Instead of being used in a method to bypass access control during its entire execution, regardless of the access control specifications of the intervening resources, trust in specified classes is explicitly acknowledged by the callee method. For example, given

```java
public class A {
    @AccessControlled(
        requires = "aPermission",
        depth = Depth.SHALLOW
    )
    public void foo() {
        new B().bar();
    }
    
    Using shallow access control should generally be considered dangerous. Hence, a mechanism should be devised to short-circuit the consequences of shallow access control. If a given method declares itself to be suspicious, its access control specification should not be turned off in the flow of a shallowly access controlled method. For instance, in

```


```
tructs = { B.class }
)
public void foo() {}
)
public class B {
    @AccessControlled(
        requires = "anotherPermission"
    )
    public void bar() {
        new A().foo();
    }
}

any call to bar() will require a principal with permission anotherPermission, as usual, but the call to foo() from within bar() will not be subject to access control, since foo() declared its trust in class B. Notice, however, that calls from within the flow of control of foo() will in general be access controlled, since trust does not propagate. This will improve even further the safety of trust relationships.

3.8 Ease of Use (12)

Zás should be easily integrated into an existing project. Indeed, if the requirements illustrated in the previous sections are fulfilled, particularly the ability to use ITDs to modularize access control specifications, little or no changes will be required in existing code.

Zás integration shall simply require
1. adding the zas.jar Java archive into the class path of the application;
2. defining a concrete aspect that extends the provided abstract aspect AccessController; and
3. adding the access control specifications either directly to the resources requiring authorization, or using ITDs concentrated in, e.g., the concrete aspect defined.

These steps are quite straightforward, with the possible exception of the definition of the concrete aspect. Access control is only possible if the current principal’s set of permissions is available. However, Zás should be as independent as possible both of the authentication mechanism used in the application, and of the roles existing in the application and their corresponding permissions. How and where to find the permissions associated with (the roles of) the current principal is not Zás’ problem. AccessController simply declares an abstract method currentPrincipalPermissions() which the concrete aspect, defined in the client code, should implement.

For example, the definition of the concrete aspect for a simple desktop application should be as simple as:
```
package pt.iscte.ci.myapp;
```

```
import pt.iscte.ci.zas.authorization.*;
public aspect MyController
    extends AccessController
    {
        private User user;
        public Set<String>
            currentPrincipalPermissions()
            {
                // get and return permissions
                // from the roles of "user"
                return new HashSet<String>(){
                    // roles
                };
            }

        before() :
            accessToControlledResources(
                AccessControlled
                )
            {
                // if necessary, authenticate user.
            }
```

4 CONCLUSIONS

A new aspect-oriented authorization package, Zás, has been proposed which leverages AspectJ to make it possible to add authorization concerns to existing applications in a simple, non-invasive way. The model used is both independent of the authentication mechanism used, and of the specific way permissions are attached to principals. Hence, while supporting RBAC, Zás is not strictly speaking RBAC-based.

Even though in its early stages of development, Zás has shown the potential of aspect-oriented approaches to authorization concerns, making them simpler to implement, support, and configure. Zás is also dynamic, allowing runtime changes to the permission requirements associated with access controlled resources. The use of Zás, which builds on a previous proposal by Laddad (Laddad, 2003), greatly reduces the scattering of authorization code and its entanglement with business code. The use of Java 5 annotations led to a model where Zás’ client code is not explicitly guiding advice introduction (Clifton and Leavens, 2002), but augmenting the expressiveness of the code by annotating it with authorization meta-information that is then taken into account by Zás’ aspects. If this is deemed unacceptable, or if it is impossible in practice, then authorization concerns can be concentrated in a single module, thus freeing business code not only from authorization-related code but also from scattered meta-information.

Zás is, in certain cases, a good alternative to JAAS: it behaves much like JAAS, though with some important limitations. For instance, unlike JAAS, it can not be used to add access control concerns to resources inside JDK classes, since AspectJ does not allow ITDs to add annotations to code inside JDK’s archives.\footnote{An interesting extension point for Zás would be the creation of an alternative to annotations to be used in such situations.}

9
However, Zás’ aim is not to replace JAAS, since Zás can be used together with JAAS-based authorization and, in a future version, Zás may even leverage JAAS authorization services.

Even though Zás is still in its infancy, we plan to revise and improve it regularly. Some possible next steps to the improvement of Zás are described next.

4.1 Further Work

In the near future we intend to improve Zás, especially taking into account the insight gained by its use in a large scale Java-based Web application.\(^\text{10}\)

Nevertheless, some points requiring further research have already been identified. Should the basic concepts of authorization be extended such that each domain object is considered a principal, with its own set of permissions and its own set of trust relationships with other objects? What is the connection between trust and the composition, aggregation, and association relations? Should a distinction be somehow drawn between query and modifier methods, in the same way we need to distinguish sets and gets in the case of attributes? How do contracts relate to authorization and access control? What does this tell us regarding the relation between the runtime permission requirements of a method and the method it overrides? What if other crosscutting concerns of the application are implemented using aspects? How do we deal with potential conflicts that may arise (including the possibility of overriding authorization controls)?

Nakajima and Tamai (\(^?\)) proposed an analysis technique to assess the coherence between authorization policies and application code. The proposal, however, assumes the authorization policies are static. How could their analysis technique be applied in the case of dynamic policies, as allowed by Zás?

ACKNOWLEDGEMENTS

Special thanks to Professor Dulce Domingos for her important suggestions and for trying to make sure we would not miss the most important authorization references.

REFERENCES


---

\(^{10}\)Namely FénixEDU®. See http://fenix-ashes.ist.utl.pt/FrontPage/.

---
ASSOCIATIVE PROGRAMMING AND MODELING:
ABSTRACTIONS OVER COLLABORATION

Bent Bruun Kristensen
Maersk Mc-Kinney Moller Institute, University of Southern Denmark, Denmark
bbkristensen@mip.sdu.dk

Keywords: Collaboration, abstraction, modeling and programming, association, concurrent and interleaved execution, activity, role.

Abstract: Associations as abstractions over collaborations are motivated and explored. Associations are seen as first class concepts at both modeling and programming levels. Associations are seen as concepts/phenomena and possess properties. Various notations for collaboration in object-oriented programming and modeling are discussed and compared to associations. Concurrent and interleaved execution of objects is described in relation to associations.

1 INTRODUCTION

Description of collaboration between some participants is less supported by existing notations and diagrams. In collaboration the participants engage in the collaboration through specific roles and the actual interaction sequence between the participants follow some rules. Existing notation and diagrams are mainly based on object-centric modeling and programming exemplified by objects, object references and remote method invocation. Examples of non object-centric notations and diagrams include the relation (May et al, 2001). This relation is an example of an abstraction over structural aspects only—interaction aspects are not covered by relations. Associations are seen as an abstraction over both structural and interaction aspects of collaboration. The association supports description at both abstract modeling and concrete programming levels. At the abstract level associations are seen as concepts and phenomena characterized by their properties. At the programming level properties of associations are expressed through language mechanisms.

Associations are inspired from a conceptual model for understanding ambient systems (May et al, 2001). Such systems have a more dynamic situation with respect to collaboration among the entities in the system. In this model we imagine tangible objects existing in habitats and collaborating with other tangible objects—and tangible objects enter and leave habitats. As part of this dynamic picture tangible objects engage in collaboration with other tangible objects—simple or complex collaborations. The notion of associations is a means of capturing planned or spontaneous collaborations between tangible objects—to conceptually understand and prescribe collaboration as abstractions over collaboration. The illustration of an ambient system included as example in this article—“the conference organizing problem”—only includes aspects of collaboration but excludes aspects of user awareness and support through knowledge of time and place, augmentation of reality by additional views, and availability and interaction with software agents and physical robots (Kristensen, 2003).

Our approach is inspired by the evolution from traditional systems (often information systems) towards ambient systems including pervasive (Burkhardt et al, 2001) and ubiquitous (Weiser, 1991) systems. Ambient systems illustrate the change from development of systems towards an understanding where systems are grown through evolution. Associations are a move from object-centric technology towards non-centric technology. For associations we distinguish between an abstract, informal and conceptual modeling level and a concrete, formal and executable programming level.
2 OBJECT-ORIENTATION

References support the relations between objects in object-oriented programming languages. In Figure 1 we illustrate the usual notions of class, object, reference and method invocation. Class C has method mc. Object Oc is an object of C. Class D has method md and a reference Rc qualified by C. Object Od is an object of D and reference Rc has the value Oc. Method md of Od can invoke method mc of Oc by Rc.mc(…).

Figure 1: Class, object, reference and method invocation.

We observe the following characteristics of this schematic example from object-oriented programming:

- The reference is statically bound to the class (and any object of the class) whereas the value of the reference varies dynamically
- The reference is qualified by a class, which determines which types of objects may be referenced by the reference
- The reference is used for different purposes (invocations of different methods from different methods)
- The use of a reference for a given purpose is separated from the reference and distributed over several method bodies

2.1 Object-Oriented Delegation

The characteristics of collaboration through object-oriented delegation include that collaboration is explicitly described and implemented in a method in Figure 3 as a1 of A1. Collaboration is initiated by Oc1.m1(…) by invoking a1 of object oA1. This approach includes some typical problems:

- Oc1 and n1 are not necessarily known to oA1 for the invocation n1(x1, x2) (“self” problem)
- A1 may be parameterized by references r2 and r3 and with some reverse reference for r

Figure 2: Illustration of object-oriented collaboration.

In a typical object-oriented collaboration as illustrated in Figure 2, method m1 of class C1 contains the following example—this collaboration example is also used in subsections 2.2 (Figure 3) and 2.3 (Figure 4):

\[
\begin{align*}
x1 &= r2.k2(...); \\
x2 &= r2.k2(...); \\
y &= n1(x1, x2); \\
r3.k3(y);
\end{align*}
\]

Figure 3: Illustration of object-oriented delegation.

2.2 Control Object/Method

The characteristics of collaboration through a control method/object include that collaboration is explicitly described and implemented in a method/object (objectification of collaboration but here only exemplified by methodification as a1 of A1 in Figure 4). Collaboration is initiated by Oc1.m1(…) by invoking a1 of object oA1. This solution includes the following problems:
The effect through $y$ on $Oc1$ must be a side effect through invocation of $r1,n1$.

$A1$ may be parameterized by references $r1,r2$ and $r3$.

Figure 4: Illustration of object-oriented control method.

3 ASSOCIATIONS

Associations represent an alternative to object-centric modeling and programming. Our associations support not only structural relationship, but also collaboration between objects. An association is described as an abstraction, it may be instantiated, and it has identity. Dynamic changing associations are supported—descriptions may be added to executing systems and objects of these may associate participating objects of the executing system.

Figure 5: Snapshots of associations.

Figure 5 illustrates dynamic creation and deletion of objects of associations through four snapshots. In (2) an object $Ax$ with roles with properties $n1$ and $n2$ is created. Object $Oc2$ of class $C2$ function as one participant in the association. In snapshot (1) no associations exist for $Oc2$. In (2) $Oc2$ is associated by means of $Ax$ with object $Oc1$ of class $C1$. In (3) the association $Ax$ no longer exists. In (4) $Oc2$ is associated by means of an object $Ay$ with roles with properties $l1$ and $l2$ with object $Oc1'$ of class $C1'$.

In UML models the main concepts are captured through class diagrams supplied with relation/association classes as fundamental model structures. In addition these models include sequence and collaboration diagrams, where the interaction of objects is modeled in terms of method invocations. This description is separated from classes and associations, and neither sequence nor collaboration diagrams are conceptualized as abstractions over collaboration. Our notion of association is an abstraction over interaction and collaboration and the actual method invocations between objects are modeled as integrated elements of the association. In addition roles played by participating objects in an association are also modeled as extensions of the objects to participate in the association. Traditionally abstractions over certain aspects of an object—as for example the collaboration of the object with other objects or the objects’ role towards other objects—are objectifications of such aspects. In this sense our notion of association is an integrated objectification of collaboration aspects and role aspects.

The association is seen as an abstraction during conceptual modelling (Madsen et al, 1993). In conceptual modeling different forms of abstraction in terms of concepts and phenomena are illustrated in (Kristensen, 2003): Classification (and exemplification) where a concept classifies a number of phenomena (which themselves exemplify the concept). Specialization (and generalization) where a more general concept generalizes a more specific concept (which itself specializes the general concept). Aggregation (and decomposition) where a whole concept describes the aggregated phenomenon of several part concepts (which themselves can be decomposed from the whole concept).

3.1 Example: paper_review

As an illustrating example, we examine the association of reviewing papers a conference—referred to as a paper_review. This association requires a certain degree of collaboration between those who are involved in it. For instance, an author will submit a paper for review, while the chairman will submit papers to reviewers who must report back. A directive describes how the association should be carried out. With the
paper_review, the directive might be carried out in distinct portions:

1. prepare_paper_review
   - author_submits_paper_to_chairman
   - chairman_submits_papers_to_reviewers
   - reviewers_submit_reports_to_chairman

2. paper_selection

3. chairman_informs_authors

The association paper_review is only one type of review that can take place. For example, a periodical_review is the review of a submitted article that takes place for a periodical; it is somewhat similar but involves an editor rather than a chairman and its selection process is different. Both paper_review and periodical_review are specialized types of review. The directive that specifies how paper_review should be carried out may also be seen as a specialization of a more general review directive:

1. prepare_review
2. carry_out_review
3. complete_review

These portions correspond to (1), (2) and (3) above (which are more specialized). The participants of these associations may also be similarly classified. For instance, all review associations involve a coordinator and an author. Thus, in a paper_review, we can refine a coordinator to be a chairman—in a periodical_review, we can refine a coordinator to become an editor.

These different types of review associations may have similar methods. For example, producing a status_report (produce a listing of the current status of the ongoing reviewing process) is something that each review association must do—a paper_review will produce a specialized type of status_report, as will a periodical_review. Finally, such associations are constituted from smaller sets of associations. For example, within the paper_review association, there is a paper_selection association to choose acceptable papers.

3.2 Execution

In addition to action directives, associations include roles to be played by participants in the collaboration. Roles are abstractions in associations. Roles may specify additional methods or may extend existing methods of an object (Kristensen, 2003). In Figure 6, R1, R2 and R3 are roles of association class X. Ax illustrates an object of class X. Ax associates objects Oc1, Oc2 and Oc3 (each playing a role R1, R2 and R3) of classes C1, C2 and C3 respectively. The method k1 of R1 illustrates an additional method—alternatively k1 may be described as an extension of the existing method n1 of C1 (similar for methods k2 and k3).

The directive of association x is executed by the respective owners of the actions among the participants of X. The notation R::k(…) means that the object playing role R executes its method k(…). Hence the collaboration is explicitly described through the directive of X e.g.:

x1 = R2::k2(…);
x2 = R2::k2(…);
y = R1::n1(x1, x2);
R3::k3(y);

An association is a description of a central abstract unit. The notation R::… is different from remote denotation, because “…” is situated in the context of R and interpreted in this context. The execution of its contribution from a directive is done by the participating object. In sequential execution, description and execution of sequencing are in terms of one execution thread only—a method in one object invokes a method of another object and one thread executes the entire invocation sequence. In multi-sequential execution, sequencing is described as several execution threads (one for each object) but is executed by one thread only. The thread switches (at language defined points) between the executions of the objects—this interleaved execution of the sequencing of objects means that only one object is executed at a given time.
4 CONCURRENCY

In general objects execute concurrently. Communication and synchronization constructs describe the interplay between such active objects. Objects have an individual action part—on instantiation, an object will immediately execute its action part and is inherently active. The description of the action part may involve the activation of methods in the object itself and (activation requests of) methods for other objects. Because the objects are active, the interaction between objects is usually coordinated by means of various forms of egocentric language mechanisms for synchronization of the execution of the life cycle of the object and method requests of/from other objects. As an example, when one object attempts a method request of another object, then the first objects must wait until the other object explicitly accepts this invocation. When the invocation is accepted the objects are synchronized and the invocation can take place.

4.1 Interleaved Execution

In associations the collaboration (including communication and synchronization between the participating objects) is described in directives of associations only. Active objects are executed in parallel (and shared data resources are typically active objects to ensure exclusive access). The association directive itself supports various ways of describing the sequencing of the collaboration including sequential, repeated, parallel, interleaved, any order executions etc. The individual action part of an object only describes its individual life cycle, i.e., no form for interaction with other objects is included. The execution of the total life cycle of an object described through several contributions in directives (of current collaborations) is an interleaved execution of its contributing parts and also interleaved execution with its individual action part. Interleaved execution for one such object of several different parts means that (at language defined points in the parts) execution will switch between the parts.

Figure 7 illustrates the mechanisms introduced. Object Ax is of association class X. Object Oc1 is a participant of class C1 and R1 denotes the role played by Oc1. The construct R1::k1(...) denotes a contribution to the directive of Ax from role R1. The object Oc1 executes its individual action part (exemplified by "... n1(...) ") interleaved with the various contributions from role R1 of directive Ax (and contributions from similar directives of associations in which Oc1 currently plays roles).

4.2 Example: paper_review

In the paper_review example, a reviewer may be actively performing other actions than those in connection with the paper_review such as researching and teaching. A teaching association between teacher, student and administration is specialized into course_teaching and supervision. Association course_teaching includes an iterative sequence of remind_students actions from teacher to students. Figure 8 illustrates how a person (with own individual action part illustrated by "... exercise(...) ") participates in both a_paper_review and a_course_teaching associations. The contribution to the person in a_paper_review includes submit_report. The contribution from a_course_teaching includes remind_student. An active person object executes exercise, submit_report and remind_student interleaved.

4.3 Specialization of Directive

Collaborations may have general directives prepared for further specialization in directives of sub-collaborations—for example the directive of the
more general association teaching may include possibilities for both course_teaching and supervision. The general directive of teaching has the form:

1. planning
2. ...
3. inner: content
4. ...
5. examination

This directive illustrates two types of specialization of collaborations: In each of 1) and 5) we illustrate virtual (part) collaborations. This means that for example planning is a virtual abstraction with some preliminary description (a directive similar to the teaching directive) and may be extended in specializations of teaching cf. virtual classes (Madsen et al, 1993). In course_teaching (as a specialization of teaching) planning may be specialized to course_session_planning. In 3) we illustrate an explicit inner construct (named as inner: content). Several such inner constructions may be specified in the directive (Kristensen, 1993), (Kristensen et al, 1996). In course_teaching the inner:content construct is specialized into a sequence of lecturing activities (for concrete courses lecturing is specified further with respect to number and content). The specialized directive is the original general directive with these two types of specializations included. The execution of a participating active object is still an interleaved execution of its individual action part and its contributions from all such specialized directives. The directive of course_teaching, specialized from the directive of teaching, includes of the sequence course_session_planning, a sequence of lecturing, and curriculum_examination of the form:

1. course_session_planning
2. ...
3. ... lecturing ...
4. ...
5. curriculum_examination

5 PROGRAMMING

We introduce association classes and objects with roles and directives in schematic programming language form. We include the paper_review example as an illustration. Finally we conclude by defining interleaved execution schematically. In general the notation ... indicates various less important or repeated parts left out of the descriptions.

5.1 Association Classes

Association classes $X_j$ include roles $R_j$ (with method $n_j$) and local associations $Y_j$ and a directive (with various characteristic ingredients to be explained later). Association object $Ax_j$ is instantiated:

```
association Xj {
  role virtual Rj for Ci {
    method nj (...) ...
  }
  ...
association virtual Yj [...] ...
directive {
  ... Yj ... inner:Ij ... Rj::nj( ...) ...
}
}

object Axj of Xj
```

Object $Oci$ of class $Ci$ enters role $Rj$ of $Axj$. A role is qualified by a class, $Ci$, meaning that only objects of this class or its subclasses, $CCi$, may enter that role. Also role $Rj$ may invoke methods of $Ci$ and $CCi$. The action part of $Ci$ illustrates various characteristic ingredients: $mi( ...)$ is an invocation of a $Ci$ method, whereas the description inner:Ii is replaced by its refined description denoted by {...} in $CCj$:

```
class Ci {
  method mi {...}
  ...
  action_part {... mi(...) ... inner:Ii ...}
}

object Oci of Ci
Oci enters Axj as Rj
class CCi extends Ci {... Ii:{...} ...}
```

Associations $Y_j$ and roles $R_j$ may be specified as virtual in order to be specialized further in specializations like $XXj$ of the enclosing association $X_j$ (for roles also the classifying class $Ci$ may be specialized as e.g. $CCi$). The directive of $XXj$ is the directive of $X_j$ where for each $I_j$ the description inner:Ij is replaced by its refined description denoted by {...} in $XXj$:

```
association XXj {
  role Rjk for CCi {...} ...
  association Yj [...] ...
  directive {
    ... Ij:{...} ...
}
```
5.2 Example: paper_review

The paper_review is presented in the language style below. The ordinary concurrent object person executes method exercise repeatedly in its action part:

```plaintext
class person {
    method exercise {...} ...
    action_part {... exercise(...) ...}
}
```

Association paper_review is a sub-association of review and specializes the directive by prepare_review, carry_out_review and complete_review. Also roles reviewer and coordinator are specialized:

```plaintext
association review {
    role virtual reviewer for person {...}
    role virtual coordinator for person {
        method submit (...) ...
    }
    role author {...}
    association virtual prepare_review {...}
    association virtual carry_out_review {...}
    association virtual complete_review {...
        directive {
            prepare_review
            carry_out_review
            complete_review
        }
    }
}
```

```plaintext
association paper_review extends review {
    role reviewer {
        method submit_report {...} ...
    }
    role chairman extends coordinator {...
    association prepare_review {
        author::submit (paper, chairman) ...
        chairman::submit (paper, reviewer) ...
        reviewer::submit_report {...
    }
    association paper_selection
        extends carry_out_review {...
    association extends complete_review {
        ... chairman::inform_author ...
    }
    directive {...
}
```

Association course_teaching is a specialized association of teaching—and refines the directive of teaching. Also the roles teacher, student and administration may be specialized—for example for teacher by adding method remind_student:

```plaintext
association teaching {
    association virtual planning {...
    association virtual examination {...
    role teacher for person {...
    role student for person {...
    role administration for organization {...
    directive {
        planning ...
        inner: content ...
        examination
    }
}
```

```plaintext
association course_teaching extends teaching {
    association course_session_planning
        extends planning {...
    association curriculum_examination
        extends examination {...
    association lecturing {
        ... teacher::remind_student(...) ...
    }
    role teacher {
        method remind_student {...
        ...
    }
    directive {
        content: {... lecturing ...
    }
}
```

Objects of course_review and paper_review are instantiated. Object John then enters the association objects a_course_teaching and a_paper_review:

```plaintext
object a_course_teaching of course_teaching
object a_paper_review of paper_review
object John of person
```

John enters a_course_teaching as teacher
John enters a_paper_review as reviewer

5.3 Interleaved Execution

Action parts of active objects are executed concurrently with directives of associations, but each action part and directive is executed sequentially:
class Ci {
    ...
    action_part {... mi(...) ...}
}

association Xj {
    role virtual Rj for Ci {...}
    ...
    directive {... Rj::nj(...) ...}
}

A given $Oci$ of $Ci$ is engaged as role $Rj$ of $Xj$ in a collection of association objects $Axj$ of $Ax$, where
1) the next individual action for $Oci$ is $mi(...)$
2) for the collection of $Ax$’s the next action to be executed for $Axj$ with $Oci$ in role $Rj$ is $Rj::nj(...)$

Interleaved execution of $Oci$ means, that exactly one out of $mi$ and the collection of $nj$’s is selected randomly and executed. For an object of a specialized class engaging in specialized associations the specialized action part and specialized directives are used.

6 RELATED APPROACHES

Notions similar to associations are available in object-oriented modeling whereas in object-oriented programming associations are implemented by means of references. Our associations support both modeling and programming (Kristensen, 2003). We include the association as a first class concept in our modeling and programming notation. In classical object-centric modeling and programming the fundamental problem is that “no object is an island” (Beck et al, 1989). In object-oriented systems an object supports encapsulation; the object is self-contained, focus is on structure instead of function and focus is on methods instead of processes. These characteristics are seen as appreciated properties of object-oriented systems, but are also essential problems because they emphasize an object-centric point of view.

Relations from (Rumbaugh, 1987) are introduced as non object-centric abstractions. In an illustrative example a relation Employment with property Salary is defined between classes Person and Company. Objects of class Person play the role of Employee and objects of class Company play the role of Employer. The relation Employment captures an abstraction, the properties of which we do not place at neither Person nor Company—the relation is between these and therefore in conflict with the intentions of the object-centric approach.

6.1 Language/Notation

Various approaches to notation for non object-centric modeling and programming include: Relations (Rumbaugh, 1987) and the corresponding associations in OMT (Rumbaugh et al, 1991) and UML (Booch et al, 1998) are object-external abstractions but these relations/associations only cover structural aspects, not collaboration. Sequence and collaboration diagrams in UML support the description of object interaction by means of method invocation, but not as abstractions and not integrated with relations/associations of objects. Complex associations (Kristensen, 1994) are object-external abstractions and support only complex structural relationships between complex, structured objects. Subject-oriented programming (Harrison et al, 1993) and subjective behavior (Kristensen 2001) support different views on objects respectively from an external and internal perspective, but not relationships between objects. Activities (Kristensen et al, 1996), (Kristensen, 1993) are abstractions over collaborations of objects, but include no support of roleification of objects participating in the collaboration. Roles (Kristensen et al, 1996), (Kristensen 1995) are abstractions over roleification of objects for various relationships of objects, but no explicit collaboration is included.

6.2 Collaboration Approaches

Design patterns (Gamma et al, 1994) capture experience of object oriented design and programming. In this approach language constructs for collaborating objects are typically simulated by patterns of objects including for example DECORATOR, OBSERVER, and MEDIATOR. Activity-based computing (ABC) (Bardram, 2005) supports mobility and cooperation in human work activities. ABC is a framework supporting a computing infrastructure to describe how to keep track of collaborative activities. The system offers a distributed, real time joint repository for activities including states, participants, communication and information. Model Driven Architecture (Zhao, 2005) is supported by the notion of roles (as a modeling paradigm) by viewing object interactions from the dimensions roles, responsibilities and collaborators. The approach yields a semantically rich model, and also a simple, elegant design that is flexible and adaptable.
ASSOCIATIVE PROGRAMMING AND MODELLING IS CHARACTERIZED BY:

- **Object-oriented programming contains object-centric descriptions, and collaboration is implicitly described only and distributed among methods of participating objects. In object-oriented methodologies alternatives exist typically only for analysis and design, but not for implementation.**

- **Associations support associative modeling and programming through abstractions over collaboration. Associations support objectification of integrated collaboration aspects and role aspects. Classification, specialization and aggregation are available.**

- **In associations directives (sequencing rules for interactions among the participating objects) are central, partial descriptions related to the participating objects. The objects execute their own contributions to the collaboration in their context. An active object participating in various associations execute contributions from the directives interleaved.**

CHALLENGES INCLUDE:

- Notation at the modeling and programming levels for creation and deletion of associations
- Entry and exit of objects in associations
- Similarities between inheritance of directives and inheritance anomaly (Matsuoka et al., 1993)

ACKNOWLEDGEMENTS

This research was supported in part by the A. P. Møller and Chastine Mc-Kinney Møller Foundation. We thank Palle Nowack, Steffen Jensen and Beata Hargesheimer for collaboration and contributions.

REFERENCES


A DECLARATIVE EXECUTABLE MODEL FOR OBJECT-BASED SYSTEMS BASED ON FUNCTIONAL DECOMPOSITION

Pierre Kelsen
Laboratory for Advanced Software Systems
University of Luxembourg
http://lassy.uni.lu

Keywords: Declarative models, executable, object-oriented programming, functional programming, software complexity, functional decomposition.

Abstract: Declarative models are a commonly used approach to deal with software complexity: by abstracting away the intricacies of the implementation these models are often easier to understand than the underlying code. Popular modeling languages such as UML can however become complex to use when modeling systems in sufficient detail.

In this paper we introduce a new declarative model, the EP-model, named after the basic entities it contains - events and properties - that possesses the following features: it has a small metamodel; it supports a graphical notation; it can represent both static and dynamic aspects of an application; finally, it allows executable models to be described by annotating model elements with code snippets. By leaving complex parts at the code level this hybrid approach achieves executability while keeping the basic modeling language simple.

1 INTRODUCTION

Abstraction is a key concept for dealing with complexity. By abstracting away details of the implementation one can construct a higher-level model that is easier to understand than the underlying code. Although the relations between successive abstraction layers are varied, a common theme is that of separating what a system does from how it is actually done. We call the approaches that rely on this distinction declarative.

An important element of a declarative approach is the language used for representing the high-level models. The de-facto standard for modeling object-oriented systems is the Unified Modeling Language (Object Management Group, 2003). The UML is a powerful language for describing systems at various levels of abstraction and from multiple viewpoints. It has a large number of diagrams available for describing systems from different perspectives, each with their own syntax and semantics. This expressiveness also means that UML is a rather large and complex language (Kobryn, 2002; Siau and Cao, 2001).

The complexity and size of the language becomes a hindrance when designing systems at a detailed level. While it is possible in principle to transform UML into an executable language (Raistrick et al., 2000) by instrumenting it with a precise Action Semantics (Alcatel et al., 2000) this results in an even bigger language. Indeed executability and simplicity seem to be conflicting goals if we judge by previous attempts. The main subject of this paper is a new executable model, the EP-model, that is based on a rather trivial observation: certain aspects of programs can be easily presented in a simple form at a declarative level while other aspects are much more difficult to capture at such a level. Our basic approach to this problem is that of leaving things that are truly complex to describe at a low level (source code) and extracting only those aspects that can easily be presented.

We now discuss the main features of EP-models and contrast them with existing approaches. The simplicity of EP-models is mainly due to the small number of concepts that they are based on: indeed the high-level metamodel can be described using only two types of entities - events and properties - and four types of relationships among those entities.

The second main feature of EP-models is their executability. Executability by itself is not a new idea (e.g., (Belina and Hogrefe, 1989; Raistrick et al., 2000)). What makes our model interesting is the fact that executability is achieved without relying on a
overly complex language for the modeling notation. Instead we propose a hybrid approach in which the model itself is unchanged but code segments annotate the various modeling elements to allow executability. A useful characteristic of our hybrid approach is the "locality" of the code segments: indeed each code snippet can only refer to the model elements that are adjacent to the element that it annotates. Clearly this locality reduces coupling since it disallows the code to access elements that it is not related to. Although there have been a few approaches to reduce coupling at the method level (the Law of Demeter (Lieberherr and Holland, 1989) is representative of such approaches) current approaches are rather low-level in the sense that they refer to an existing class structure. On the other hand the EP-models provide a "sandboxing" approach for code that is situated at a higher semantic level.

Finally, EP-models model both static and dynamic aspects of a system in a single diagram. On the other hand UML separates static and dynamic aspects into different diagrams. One reason for this difference lies in the fact that while UML is largely grounded in the object-oriented paradigm our model combines ideas from both object-oriented and functional programming: it borrows the notion of state from object-oriented paradigms but also allows functions without side-effects that are decomposed over the state. We remark that the idea of combining the functional and object-oriented paradigms is not new but most attempts have focused so far at the level of programming language design (e.g., (Hughes and Sparud, 1995; Remy and Vouillon, 1997; Odersky and Wadler, 1997)).

3 THE STATIC VIEW: LOCAL PROPERTIES AND THE SYSTEM STATE

The static structure of an EP-system is given by a set of local properties in each model. (A second class of properties named query properties will be introduced in section 6). Local properties have a name and a type. We shall assume that no two properties in the same model have the same name. The type has a name and an associated set of values. This type can be either internal or external: an internal type is given by another model in the EP-system. Examples of external types are the built-in types of a programming language or a class in a class library; external types are not represented by EP-models. A property is either single-valued or multi-valued. We call multivalued properties also collection properties.

Example 1 We name the EP-model for representing a flash card FlashCard. This model contains two properties, named question and answer, of type java.lang.String, an external Java type. Another example is the Main EP-model representing the main window of the FlashCards application. The addButton and quizButton properties are properties of an external type (SWING components). The other properties of the Main model - list, cardDialog, quizDialog, flashCards - have an internal type represented by an EP-model. We note that the flashCards property is a collection property of type FlashCard - this property refers to the collection of flash cards entered by the user.

When an EP-model executes, it goes through a series of system states. Informally, a system state is a set of instances, each instance belonging to some model and assigning concrete values to the local properties in that model.

For a more formal definition of a system state:

Definition 1 A valuation for a model $M$ is a function that assigns to each local property $p$ in $M$ a value of the type of $p$.

Definition 2 An instance of model $M$ is a triplet $(M, id, \phi)$ where $M$ is a model, id is a name for the instance and $\phi$ is a valuation for $M$. We call $\phi(p)$ the value of (local) property $p$ in $M$ on this instance.

Definition 3 A system state is a set of instances.

Condition In any system state the id’s of the instances are unique.
4 EVENTS, THE TRANSFORMATION MAPPING AND CENTERED FUNCTIONS

External triggers that modify the current system state are represented in a model by local events. (Another class of events - remote events - will be presented in a later section.) A local event has a name, a type and a source. We shall assume that no two events in the same model have the same name.

The event type is platform-specific: in Java an event type is a pair \((l, m)\) where \(l\) is a listener interface and \(m\) a method of this interface.

The source of an event is a local property in the model that contains the event.

Example 2 In the Main model we have two local events add and quiz. The add event represents pressing the add button and the quiz event occurs when we press the quiz button. Both events have as type \((\text{java.awt.event.ActionListener, actionPerformed})\). The source of the add event is the addButton property of the Main model and the source of the quiz event is the quizButton property of the quiz event.

Definition 4 We say that a local event occurs on an instance \(x\) if an (external) event of the given type occurs on the object referred to by the source of the local event. In this case we also say that instance \(x\) is the locus of the local event.

Example 3 When we press the add button in the main window, an event occurs on the Main instance; this instance is then the locus of this event.

Definition 5 When a local event occurs on instance \(x\) of the current system state, then the current state is replaced by a new state which we call the result state.

Definition 6 For a given local event \(e\) the mapping that associates with each system state and instance of this state on which \(e\) occurs a result state is called the transformation mapping for \(e\) and is denoted by \(F_e\).

Mathematically we describe transformation mappings using centered functions.

Definition 7 A centered state is a pair \((s, x)\) where \(s\) is a system state and \(x\) is an instance of \(s\). We call \(x\) the center of the centered state.

Notation We also use \(s(x)\) to denote a system state \(s\) centered at \(x\).

Definition 8 A centered function is a function whose domain is a set of centered states (for the given EP-system).

We may view the transformation mapping \(F_e\) as a centered function that maps the current state centered at the locus of the event to the result state.

The transformation mapping completely describes the dynamic behavior of an EP-system. Indeed if the EP-system expresses the transformation mapping precisely, then the EP-system is executable. The remainder of this paper is essentially looking at the question of how to best represent centered function \(F_e\) at the level of the EP-models.

5 BICENTERED FUNCTIONS

To represent the transformation mapping, we shall decompose it into simpler functions. First we need to define the effect an event has on a system state.

Definition 9 A local event \(e\) affects a local property \(p\) if for some system state the value of this property is changed on some instance of this state when the event occurs. In this case we also say that the local event affects property \(p\) on this instance.

Example 4 The effect of the quiz event in the Main model is to show the QuizDialog, to set the questionField (a text field) to the first question and to set the index property indicating the position of the current card among the stack of flash cards. Thus the quiz event affects the visible and index properties of QuizDialog.

To fully describe a local event \(e\), it suffices to specify the effect of \(e\) on each local property affected by \(e\). The effect of \(e\) on property \(p\) can be expressed by the function that returns the new value of property \(p\) on an instance of the result state after \(e\) occurs on the current state; we denote this function by \(F_{e,p}\).

Example 5 Let \(e\) denote the quiz event in the Main model and let \(p\) stand for the visible property in the QuizDialog model. Then \(F_{e,p}\) represents the new value of the visible property when the quiz event occurs; in this case \(F_{e,p} = \text{true}\).

The value of \(F_{e,p}\) depends on two centered states:

- the current state \(s(x)\) centered at the locus of local event \(e\), i.e., at the instance where \(e\) occurs
- the result state centered at an instance at which we are evaluating the new value of \(p\)

This dual dependency motivates the next definition.

Definition 10 A bicentered function is a function of the form \(f(s(x), s'(y))\) where \(s(x)\) and \(s'(y)\) are two system states centered at instances \(x\) and \(y\), respectively.

We note that function \(F_{e,p}\) is a special bicentered function where the second argument state is the result state.

The centered function \(F_e\) is fully specified by the functions \(F_{e,p}\), where \(p\) ranges over all properties affected by \(e\). We have thus reduced the problem of decomposing the centered transformation mapping \(F_e\)
into that of decomposing the related bicentered functions $F_{c,p}$. Before we address the decomposition of bicentered functions, we explain how to decompose centered functions since they will be used in the decomposition of the bicentered functions.

6 DECOMPOSING CENTERED FUNCTIONS USING PROPERTY GRAPHS

In this section we shall describe how to decompose centered functions and how to represent this decomposition in EP-models.

The computation of a centered function will be based on the decomposition of this function into “simpler” functions. Each centered function is represented at the model level by a query property. Just like local properties query properties have a name and a type. Local properties and query properties together make up the set of properties of an EP-system. To decompose the query property, we first describe which values a query property depends on. This is done by defining for each query property a property graph.

Definition 11 The property graph for a query property $q$ is defined as follows: the set of nodes is a set of properties that contains property $q$ and other local or query properties. There are three types of edges: local edges, forward edges and inverse edges. A local edge is given by a pair $(p_1, p_2)$ of properties in the same model. Forward edges and inverse edges are labeled by a property $p$ which we call the link property; for forward properties the link property is a property in the model of $p_1$ whose type is a model containing $p_2$ while for inverse properties the link property belongs to the model of $p_2$ and its type is a model containing $p_1$. The link property is a local property or a query property.

Intuitively, a local edge represents a dependency of two properties on the same instance while forward and inverse edges represent a dependency between two properties on two separate instances connected by the link property $p$.

At the model level we represent the property graph by adding a parent relationship link from a query property to each of its children properties. The parent relationship has two attributes: the link property (undefined for local edges) and type (local/forward/inverse).

Example 6 Figure 1 shows the property graph for the query property nextIndex in the QuizDialog model: this query property computes the index of the next card to be displayed in the quiz dialog. The property graph contains two local edges and one inverse edge (having link property quizDialog). The flashCards and index properties are local properties.

We add a code snippet to each query property that computes the value of the query property in terms of the values of children properties.

Example 7 The code snippet that computes the value of property nextIndex is given below. Note that it only uses values of properties that are children of itself in the event graph (see figure 1).

```java
if (index<cards.size()-1)
    result = index+1;
else
    result = 0;
```

7 DECOMPOSING BICENTERED FUNCTIONS USING EVENT GRAPHS

To decompose the $F_{c,p}$ functions, we will need to precisely define what instances in the result state are affected by an event. This will be done by associating with each local event an event graph.

To define the event graph for a local event, we first add to each model a set of remote events. Remote events have a name but unlike local events they do not have a type and source attribute. Local events and remote events together make up the set of events of an EP-system. We may think of remote events as the representatives of a local event in other EP-models.

Definition 12 The nodes of the event graph of a local event comprise the local event as well as a set of remote events. The edges of the event graph are either forward, inverse or local edges. A forward edge $(e_i, e_j)$ is labeled by a property $p$ in the model of $e_i$; $e_j$ must be an event in the model that is the type of $p$. The forward edge is denoted by $e_i \rightarrow_p e_j$. An inverse edge $(e_i, e_j)$ is labeled by a property $p$ in the model of $e_j$; $e_i$ must belong to the model that is the type of $p$. The inverse edge is denoted by $e_i \leftarrow_p e_j$. A local edge $(e_i, e_j)$ connects two events in the same model and does not carry a label; it is denoted by $e_i \leftrightarrow e_j$. For forward and inverse edges we call property $p$ the link property. The link property is a local property or a query property. We call the edges in the event graph also event links.
Definition 13 A parametrization of an event graph $G_e$ is given by

- attaching to each event of $G_e$ a set of centered functions represented by query properties
- assigning to each remote event of $G_e$ a set of parameters, where each parameter has a name and a type
- assigning to each event link $l$ and to each parameter $g$ in the target event of $l$ a function $F_{g,l}$ that expresses the parameter $g$ in terms of parameters and centered functions $f_i$ at the source of $l$: $g = F_{g,l}(f_1, \ldots, f_k)$.

We now describe how the parametrization is represented at the model level. We may view a local property as a very simple centered function that returns the value at the center of its argument state. We represent more general centered functions in EP-models by query properties (see previous section).

We represent the attachment of a centered function to an event by introducing a feeds relation from the query property to the event. Each non-local event in an EP-model has a set of parameters. The function $F_{g,l}$ is represented by attaching a code snippet for parameter $g$ to the event link from the source to the target event.

We have now all the pieces together for implementing a local event $e$ at the level of an EP-system: decompose the transformation mapping $F_e$ into the bi-centered functions $F_{e,p}$, one for each property $p$ affected by $e$. For each function $F_{e,p}$ create an event graph with a parametrization that provides a functional decomposition of $F_{e,p}$. Note that this may entail creating new query properties feeding remote events. Finally decompose these query properties using property graphs. Repeating these steps for each local event will result in an executable EP-system that represents the final application. For a more detailed description and additional examples the reader is referred to (Kelsen, 2006).

8 APPLICATIONS OF EP-SYSTEMS

We see three directions for future work that correspond to potential applications of EP-models.

1. Modeling applications: as we have seen in this paper we can use EP-models to model simple applications. Because EP-models are executable the EP-system in fact constitutes the application: no additional code is needed. Of course in practice one would write a code generator for efficient execution. To prove the feasibility of this approach we have developed a tool (Glodt and Kelsen, 2006) that provides a visual environment for designing EP-systems: the tool is implemented as an Eclipse plug-in that supports editing and executing EP-models with rule-based background code generation. It is not clear yet whether EP-models are a reasonable approach for modeling large applications: to answer this question, we are planning to develop such an application using our tool. In any case EP-models would supplement rather than replace existing UML models: indeed many UML artifacts such as use cases and deployment diagrams could supplement EP-models by providing high-level views of the application and also describing aspects not represented by our models.

2. Mastering software complexity: EP-models have a number of features that may help in controlling the complexity of the resulting system: first EP-systems exhibit locality because code snippets may only depend on values that are located on “adjacent” elements in the EP-system. We have implemented (Glodt and Kelsen, 2006) this locality using a sandbox model: the sandbox for a code snippet only contains the values that are accessible by this code in the model. This locality should help in reducing coupling ((Stevens et al., 1999)) in the resulting application. Second the models provide
facilities for comprehending the dynamic behavior of an EP-system: we can understand the effect of an event on the system by following the edges of the event tree. Similarly data dependencies can be quickly discovered with the help of property trees.

3. A laboratory for testing object-oriented methods and concepts: since our models provide a restricted environment for describing the static and dynamic aspects of an application, they should be easier to analyze and can be used as a testbed for developing mathematical models that may carry over to more unrestricted environments. For example techniques such as design patterns (Gamma et al., 1995) or refactoring (Fowler, 1999) could be examined in these more restricted models. This could potentially provide a more rigorous basis for these techniques that could carry over at least in part to more traditional software programs. Another benefit of trying out these techniques on EP-models is of course their potential to make the EP-modeling process more effective.

9 CONCLUSIONS

We have presented a declarative model, named the EP-model. EP-models are based on a small metamodel comprising two types of entities, events and properties, and four binary relationships between those entities.

EP-models are executable; executability is achieved by associating code snippets with entities and relationships. These code snippets compute functions without side-effects. This hybrid approach allows one to keep the basic modeling language simple by leaving complex parts at the code level. The code snippets obey a locality constraint: they can only use values connected with neighboring modeling elements. This reduces the amount of coupling in the resulting application.

EP-models combine static and dynamic aspects of a system in a single diagram. They combine the notion of state from object-oriented programming with the notion of functional decomposition from functional programming.

Future work will examine

- whether EP-models can be used to model applications of a realistic size and what the advantages/disadvantages are over existing UML-based methods;
- whether EP-models can be used as a cleanroom for testing object-oriented ideas and concepts. As an example we plan to study refactoring and design patterns in the context of these models. Because of the simple structure and executability of these models, such a study could provide a more rigorous basis for some of these techniques which could then possibly be carried over to more traditional software programs.

REFERENCES


Posters
AVOIDING TWO-LEVEL SYSTEMS: USING A TEXTUAL ENVIRONMENT TO ADDRESS CROSS-CUTTING CONCERNS

David Greaves
University of Cambridge, Computer Laboratory
Cambridge, UK
Email: david.greaves@cl.cam.ac.uk

Keywords: Aspect Oriented Programming, Meta-Programming, Textual Environment, Interceptor Function.

Abstract: We believe that, owing to the paucity of textual facilities in contemporary HLLs (high-level languages), large software systems frequently require an additional level of meta-programming to sufficiently address their cross-cutting concerns. A programming team can either implement its system by both writing the main application in a slightly customised language and the corresponding customised compiler for it, or it can use a macro pre-processor to provide the remaining cross-cutting requirements not found in the chosen HLL. With either method, a two-level system arises. This paper argues that textual macro-programming is an important cross-cutting medium, that existing proposals for sets of pre-defined AOP (aspect-oriented programming) join-points are overly constrictive and that a generalised meta-programming facility, based on a textual environment should instead be directly embedded in HLLs. The paper presents the semantics of the main additions required in an HLL designed with this feature. We recommend that the textual features must be compiled out as the reference semantics would generally be too inefficient if naively interpreted.

1 INTRODUCTION

Although the term is relatively new, cross-cutting requirements have always been found in large software projects, and have been met in various ways, aligned with various cutting/joining axes that we list in the next paragraph. This paper emphasises one particular cutting axis, that of the textual structure of the source file. We first provide a list of the main cutting axes, then list ways in which the textual axis is exploited by the C/C++ pre-processor and speak of alternatives when a pre-processor is not used. Finally we introduce our own textual-environment concept to HLL (high-level language) compilation and evaluate it in terms of how it addresses the facilities otherwise provided by a pre-processor or customised compiler.

By means of introduction, we list the common facilities provided for cross-cutting found in HLLs, starting with the most basic. We define a ‘cross-cutting’ aspect of a language to be any mechanism within the language that provides a link from one part of the program to another. It reduces the effective diameter of the program by increasing the dimensionality of the interconnectivity.

Cross cutting axes we identify are:

1. Shared Global Variables. Shared variables are the most ancient and basic cross-cutting facility, according to our definition thereof. Although obvious, we list them for completeness.

2. Thread Dynamics. A thread weaves between sub-routines, often held in separate textual files. This is one of the most distinguishing features of

\[\text{Figure 1: Cross-cutting aspects feeding either into the upper or the lower crust of the sandwich that contains the main application code.}\]

1. \textit{Shared Global Variables.} Shared variables are the most ancient and basic cross-cutting facility, according to our definition thereof. Although obvious, we list them for completeness.

2. \textit{Thread Dynamics.} A thread weaves between sub-routines, often held in separate textual files. This is one of the most distinguishing features of

\[\text{Strangely, in the hardware description languages VHDL and Verilog, threads may not move between modules. This is one of the most distinguishing features of}\]
3. **Static and Dynamic Reference Environments**. Apart from access to global and dynamically allocated local variables, prograrming languages with dynamic-free variables, such as Pascal, OCAML and dialects of Algol, provide function closures where a function can make direct reference to values in its closing environment, even when it has been passed off for remote execution as an upcall (see following note).

4. **Computed Branches and Upcalls**. Languages that enable function pointers to be stored in variables, which is all modern languages, enable dynamic dispatch and remote invocation of these functions. Where one component stores an entry in the data-structure of another (often providing a so-called up-call), this is a cross-cutting feature.

5. **Object Static and Dynamic Hierarchy**. The static module inheritance graph of an OO language and the dynamic, actual instantiation of an object mesh at run time are both forms of cross-cutting. We note that Java decided not to provide access to dynamic function variables that are free in a method because it instead provides access to the fields in the surrounding object that may be dynamically allocated almost as easily. Programmers used to SML, OCAML, Haskell and so on often find this a nuisance, but the overhead of providing both a static chain and an object context, with the former not likely to be used by contemporary mainstream programmers, was seen as too great by the Java designers; hence they traded one form of cross-cutting for another.

6. **Long Jumps and Exceptions**. Long jumps and exceptions form another cross-cutting aspect (by our definition) whose usefulness is well proven.

7. **Macro Pre-Processing**. The C pre-processor is used to provide a whole gamut of cross-cutting facilities, which we list separately below. Pre-processing is often deprecated because it is crude, being not type-safe and offering the potential to make a program unreadable. This paper will hold that pre-processing should be replaced with a well-designed, yet simple, ‘meta-programming’ facility that is a primary part of any compiled HLL.

8. **Constructors, Access Functions and Overloading**. OO languages allow the user to insert their own code at points where abstract datatypes are created, read or otherwise operated on, by writing constructors and methods bound to overloade operators. These are useful joinpoints.

9. **Templates and Generics**. Some would argue that C++ templates and other similar HLL generics are these languages that enforces a totally different programming style from that used in all (other) software. artifacts to overcome antiquated, non-parametric, type systems, and would suggest using HM (Milner, 1978) typing instead. However, in whatever way the type system works, the facility to insert additional code in the template libraries provides a form of cross-cutting. Provision of some form of polymorphism, even just through ‘void *’ casts, is a required cross-cutting form for any HLL used in a large system.

We assert that the tacit motivation for AOP (aspect-oriented programming) is that those languages that do not normally use a pre-processor are restricted because the remaining parts of the above list are insufficiently expressive. The sandwich diagram, figure 1, shows that some cross-cutting requirements can be met by the facilities of the HLL *per se*, whereas the remainder are implemented using some form of meta-programming. Two forms are shown. Either a pre-processor is used, by which we imply to include the operations of this nature performed by an IDE (integrated development environment). This is the upper crust approach. Otherwise, customisation of the compiler/interpreter is needed. This is the lower-crust approach. We believe that only one crust is needed to provide sufficient additional cross-cutting. In our terms, Aspect-J (Kiczales et al., 2001) is a lower-crust approach, where modification of the compiler serves this purpose and the modifications are sufficiently flexible to provide fairly generic meta-programming. One can also argue that the lower crust represents a frequent, major motivation for developing specialised HLL’s, such as database languages. In the author’s personal experience, where a project team has worked on an ML program some 100,000 lines long, occasional customisations to the ML compiler have proved invaluable when certain cross-cutting requirements have arisen. These all fall into one of the application categories listed below.

The C/C++ preprocessor embodies many individual functions and is defined in terms of multiple passes of the source files. However, a small and well-known core of operations is all that is commonly required. Although most readers of this paper will be well-familiar with the C preprocessor and its typical uses, it is worth listing those specific uses here, so that the reader can consider our assertion, in terms of each use, that AOP has been an attempt to re-provide these facilities when a pre-processor is not routinely used or is deprecated, or the run-time system cannot be customized. The list also serves as the basis for our evaluation criteria in the results section of this paper. The C/C++ pre-processor is commonly used for the following (cross-cutting) functions. We note that the majority2 of these functions can be provided using the

2When we say the ‘majority’ we could have put ‘all’ because the residual language is, of course, Turing complete.
other major cross-cutting forms listed above, such as sending a thread to a method or carrying extra parameters into a function, but that the alternative would be unnecessarily expensive compared with the pre-processing function.

- **Tie-Offs.** A tie-off is the permanent setting of a variable to a constant value. For instance, hard-wiring the name of a directory path or the IP address of a primary server.

- **Conditional Compilation.** From the point of view of this paper, conditional compilation is a tie-off to the guard expression of a conditional construct.

- **Textual Inclusion and Access to Textual Context.** The C pre-processor allows one file to include another and enables the name and line number of the current file to be accessed, using `_FILE_` and `_LINE_`. Newer HLLs, like Python and Java, provide reflection APIs that allow much more information about the textual structure of the program source to be read off. Our textual environment, introduced later, relies on this information being available.

- **Assertions.** Pre-processor assertions generally require both conditional compilation (for rendering a speed-enhanced version) and a long jump or exception mechanism. Provided these two cross-cutting axes exist, conventional assertions can be implemented. If we have textual inclusion and access to textual context, they can be placed in their own library and print the details of their caller.

- **Visualisation, Logging, Coverage Monitoring, etc..** The conditional compilation aspect of the pre-processor enables logging to be turned on and off with global switches, and access to textual context enables C/C++ macros to be conditional, as well as providing vital information to index the recorded data. There are many variations of logging, including visualisation of program resource consumption and code coverage testing. Logging was the example cross-cutting concern addressed in several AOP papers (Kiczales et al., 2001).

- **Memory Allocation and Tracking.** The pre-processor is often used to implement a ‘new’ macro in C, overcoming a historical deficiency and the tracking functions just require a cross-cutting logger.

- **Watchpoints.** Using the pre-processor, it is easy to provide breakpointing when a thread reaches a particular line of code; watching for a variable to attain a certain value requires that all writes are ensconced inside unpleasant macro calls; checking for the formation of a particular pattern in a data structure is not at all easy. The customised compiler approach is perhaps the easiest conventional way to watch for the latter, when available; otherwise specialist hardware techniques are used, which are beyond the scope of this paper.

- **Accessor Functions for Opaque Data-Structures.** Inter-procedure call optimisation has replaced the use of *in-lined* macros as the best way to access otherwise opaque data-structures. This use is obsolescent.

- **Inter-language Calls and Miscellaneous Application-Specific Uses.** There are many other applications for the pre-processor, but we assert that the remaining uses can be regarded as application-specific rather than cross-cutting. These include persistence, scheduling, and a whole host of library, operating-system and inter-language calls. The majority of these uses cannot be coded in the original HLL, or certainly require deferred linking, and hence are not cross-cutting aspects of the current application.

A textual (or typographical) technique known to all mathematicians is the distributive law. We assert it can be helpful in programming. For instance, if $g()$ does not produce side effects referenced by $f()$, and vice versa, then

\[
(f(\text{if } g() \text{ then } A \text{ else } B))
\]

can be rewritten as

\[
\text{if } g() \text{ then } f(A) \text{ else } f(B).
\]

Our assertion is that the process of ‘folding in’ is a required form of cross-cutting in large software systems, where the desire is to apply $f()$ at one point and have it executed at many, textually lower, points. Our approach is to pass items such as $f()$, as well as tie-offs that might effect various functions like $g()$, down through the textual structure of the program to the leaves where they will act.

## 2 EMBEDDED PRE-PROCESSOR

In this section we define what is essentially a powerful, embedded pre-processor. This is specifically designed to serve the cross-cutting aspects that have been met with a second level, that of macro-processing, as identified in the previous section.

We assume the abstract syntax tree of our HLL is very typical, like the following:
Figure 2: The clause for Function Apply taken from our toy interpreter. For clarity, it is first shown without the entry and exit calls to the \textit{mf} accessor joinpoint. The triple returned is the result, the modified environment $\sigma'$ and the modified textual environment, $T'$. 

Rather than presenting only the denotational semantics for the embedded pre-processor, we alternate the presentation by giving fragments of SML from a toy implementation of the interpreter. Hence, we write 'eval(ast, sigma, text)' instead of [\textit{ast}](\sigma, T'), where ast is a fragment of abstract syntax tree, $\sigma$ is an association list for the environment, mapping variables to values, and $T'$ is our new textual environment. In the compiler, as opposed to the interpreter, $\sigma$ is a symbol table mapping variables to run-time store locations. Additional arguments would be needed to support either dynamic free variables and/or the OO 'this' current object pointer, but these are bookwork and omitted for clarity. The demonstrator in SML can be downloaded from the following URL: http://www.cl.cam.ac.uk/users/djg/aspectsdemo.

A full implementation of the textual environment, $T'$, would be too long to present in this paper, and its fine detail is not very important. The significant aspects are:

1. It is initialised as a set of bindings/tie-offs by command line flags, such as the -D flag used in C/C++, the source file path, using URI etc. and from other compile-time environment settings.

2. It is temporarily augmented, in the style of a LIFO stack, by an explicit 'meta' construct as well as by entry to each nested block or textual inclusion.

3. A set of access functions and predicates provided as natives in the HLL are able to extract values and test properties of $T'$.

4. Names of variables and functions appearing elsewhere in the source program can be stored in $T'$ to produce special behaviour where they occur.

The simplest access function would be the direct occurrence in an HLL expression of the name of a textual variable, bound only in $T'$. Variables are looked up in $T'$ before $\sigma$ to give precedence to tie-offs. To avoid over-pollution of the expression namespace, specific textual values should be extracted from $T'$ with an HLL primitive such as '.T()'. For example, \texttt{.T[name]}(\sigma, T') = \texttt{name(T')}, where \texttt{name} is one of many possible builtin accessor functions for textual context, e.g. one that retrieves the clos-
AVOIDING TWO-LEVEL SYSTEMS: USING A TEXTUAL ENVIRONMENT TO ADDRESS CROSS-CUTTING CONCERNS

est textually-surrounding basic block name. Others would access line number and file and directory name.

Conditional compilation is implemented using the language’s conventional conditional constructs, such as ‘if’, ‘case’ and ‘?:’, where the guard expression makes access to Te.

The sequence rule, typically denoted in the concrete syntax with semicolon, is augmented by the HLL parser to supply additional context information such as the line number for each sequence operator. We denote the meta information at the semicolon by the suffix $\sigma$: $\langle e_1; e_2 \rangle^\sigma = [\langle e_1 \rangle^\sigma, \sigma, \langle e_2 \rangle^\sigma]

The sequence rule evaluates $e_2$ using both environments returned from $e_1$. Imperative basic blocks (generally denoted with braces or begin/end) are built out of the sequence rule in the normal way, except that the finally returned textual environment is the initial one that acted at the start of the block, thereby deleting bindings created inside the block.

User tie-offs (bindings) can be locally introduced into Te with the HLL ‘meta’ construct that augments the textual context for the remainder of the current basic block.

$\langle \text{meta}(v, e) \rangle^\sigma = (\bot, \sigma, (v, e)^\sigma) :: \text{Te}$

Note that $\sigma$ is ignored when evaluating $e$ since it is not known until runtime in the compiler version.

The basic function application step, when using the textual environment, is presented in the top part of figure 2. This implements the procedure and function call operation, using call-by-value. The actual parameters are evaluated in order, leading to successive new bindings in $\sigma$, as well as any other side effects, before the body is evaluated in the final $\sigma$, denoted $s2$.4 Note that the textual environment for evaluating the body, $t'$, is that from the function definition, rather than that of the caller. In an interpreter, it would be helpful to provide access to aspects of the caller’s textual environment, but this would add considerable run-time overheads, if supported for separately-compiled modules.

4With the given simple code, the binding of the bound variable is left in the returned environment, $s3$, but ideally this would not be the case in reality, such as our own reference implementation.

5Where dynamic-free variables are used, the eval of $f$ will return a closure to augment sigma during the eval of the body. This form of cross-cutting should be considered to be applied to all of our fragments, but we do not show it for clarity.

To provide logging and accessor functions for formal parameter, variable and field access operations we allow user-defined interceptor functions to be registered in Te, associated with any variable, that are called when that variable is read or written (second argument is a 1 for a write). The execution semantics for this are where $f : \alpha \rightarrow \text{int} \rightarrow \alpha$ is associated with variable $b$, then, on a right-hand side, $[b]_{(\sigma, \text{Te})}$ is replaced with $f([b]_{(\sigma, \text{Te})}, 0)$ and, on the left-hand side $[b := e]_{(\sigma, \text{Te})}$ is replaced with $b := f([e]_{(\sigma, \text{Te})}, 1)$. This is a more general form of the tie-off, but can be defined with the same concrete syntax: namely $\text{meta}(f, e)$. Locality of operation is controlled both by the restricted scope of the meta construct and the ability of the accessor functions to test Te to gate their behaviour. To achieve global control, starting values are established in Te on the compiler command line or via the IDE.

Interceptor functions also serve at the function call and return join points. They operate before the call, but after evaluation of the arguments, or at the return and on the return value. The meta construct can again be used to set up the desired action, associating a number of user constant tie-offs or user interceptor functions with the locally enclosing function or any named function called while the subsequent textual environment holds.

Specifically, ‘$\text{meta}(\text{af, mf})$; $e2$’ causes all calls to $\text{af}$ within $e2$ and any subsequent sequential commands in the same basic block to have their return value passed through function $\text{mf}$, for logging or tie-off. The second argument to $\text{mf}$ is a 1 on the return stroke. On the before-call stroke, the second argument is a 0 and its return value is ignored. For callee side interpolation, $\text{af}$ is replaced in the meta statement with null or some other token to signify the current function in all functions defined within block starting with $e2$.

The implementation of the caller’s side interceptions is illustrated in the lower part of figure 2, although, for brevity, only one interceptor function, $\text{mf}$, is retrieved per function call instance. After evaluating the actual arguments, Te is searched by the caller for a definition pertaining to the called function. In an efficient interpreter, the search result would be cached, using whatever technique is already deployed for optimising branches, whereas for compilation the search is only made once anyway. Also shown in the code is a helpful facility to intercept the caller’s arguments in either the pre- or post-call joinpoint. It would be handy to access these during the execution of $\text{mf}$ using the formal names they are bound to in the callee, but this cannot be done if the callee is compiled separately or a computed branch is used where different formal identifiers are used in different destinations. Therefore, as a fallback, hardwired, stylised identifiers are always provided, such as $\bot$, $\_2$, to ac-
cess the actuals by positional index.

The implementation of the callee-side joinpoints is similar. As mentioned, a special token, such as the empty string, should be passed as the first argument to the `meta` statement to register an interceptor function for entry and exit to the currently textually enclosing procedure or function definition.

Another useful feature is for `Te` to contain a handle on the stack pointer so that calls can be associated with their returns. The actual stack pointer is easily mapped to a simple integer on most machines, and the integer can then be accessed as `‘sp’`. Although it is intended that only the relative values of the integer have meaning, bugs that arise from use of the actual values will tend to be machine-dependent. Compile-time static analysis can ensure that no reliance of actual values is used, but we have not implemented that.

Although we have only implemented an interpreter for the textual environment feature, we have spoken of the constraints and benefits arising from the compiled implementation. Compilation is certainly our intended medium, not least for efficiency. Standard techniques for converting an interpreter to a compiler (Futamura, 1999) are not made less practical by our approach.

Small-scale trials would be the best form of evaluation of the presented work, but we have not had a chance to start them. Nonetheless, if the reader now scans again the list of applications addressed by the two-level system, we believe it is more-or-less obvious that use of our textual environment, which is automatically augmented with meta-information by the command line, IDE, textual inclusion and named block and sequencing operators, can adequately address the application list.

## 3 Conclusion

This paper has presented an original and comprehensive definition of weaving and cross-cutting methods and applications. We asserted that the main uses are conventionally met using a single level of meta-programming (not a new assertion (Volder, 1999)). Two possible levels were presented: pre-processor and customised HLL. We used the C/C++ pre-processor as our main example owing to it being widely familiar. We then presented the essence of a general-purpose, single-level compilation technique that provides all of the methods and applications we found in the pre-processor. Our solution requires almost no additional syntax in a concrete implementation and is therefore claimed to be superior to other proposals.

An extension to our system would allow writes as well as reads to the textual environment. This would facilitate storage of meta-data needed for emerging dynamic-binding applications based on reflection APIs and so on.

Please note that although we have used a functional language (SML) to express the main evaluation function, our approach applies equally to imperative and functional target languages.

## References


Software Engineering
Full Papers
DEVELOPING A CONFIGURATION MANAGEMENT MODEL FOR USE IN THE MEDICAL DEVICE INDUSTRY

Fergal McCaffery
Lero – The Irish Software Engineering Research Centre, University of Limerick, Ireland
Fergal.McCaffery@dkit.ie

Rory O’Connor
School of Computing, Dublin City University, Dublin, Ireland
roconnor@computing.dcu.ie

Gerry Coleman
Department of Computing, Dundalk Institute of Technology, Dundalk, Ireland
gerry.coleman@dkit.ie

Keywords: Configuration Management, Medical device, Software Process Improvement, CMMI.

Abstract: This paper outlines the development of a Configuration Management model for the MEDical device software industry (CMMED). The paper details how medical device regulations associated with Configuration Management (CM) may be satisfied by adopting less than half of the practices from the CM process area of the Capability Maturity Model Integration (CMMI). It also investigates how the CMMI CM process area may be extended with additional practices that are outside the remit of the CMMI, but are required in order to satisfy medical device regulatory guidelines.

1 INTRODUCTION

Software is becoming an increasingly important aspect of medical devices and medical device regulation. Medical devices can only be marketed if compliance and approval from the appropriate regulatory bodies of the Food and Drug Administration (FDA Regulations, 2002), and the European Commission under its Medical Device Directives (European Council Directive, 1993) is achieved. Medical device companies must produce a design history file detailing the software components and processes undertaken in the development of their medical devices. Due to the safety-critical nature of medical device software it is important that a highly efficient CM process is in place within medical device companies.

CM is the discipline of coordinating software development and controlling the change and evolution of software products and components (Ghezzi et al, 2003). It involves the unique identification, controlled storage, change control, and status reporting of selected intermediate work products, product components and products during the life of a system’ (Jonassen-Hass, 2002). Such CM procedures are needed to manage the vast number of elements (source code, documentation, change requests, etc) that are created and updated over the lifetime of a software system.

For many software companies, who report CM problems, CM is the first major process weakness that they are required to address. For example, as the company expands, it must fulfil the task of acquiring new customers whilst satisfying the demands of existing customers. Often these demands include product customisations which many young companies, lacking reliable revenue streams, do not feel they can ignore. In many situations this results in companies having to support multiple code bases and product versions with very limited resources. Ultimately, a detailed CM process is the only way this problem can be solved.

A study of a small Danish software firm shows how it was forced to review the number of products it developed, and the amount of work it accepted, because of CM difficulties (Baskerville and Pries-Hje, 1999). But CM is equally important in large software companies as a case study of Netscape and
Microsoft’s development practices shows (Cusumano and Yoffie, 1999). Therefore, in a software company or department without CM to control product development, there is no process to assess and no basis for measurement (Fayad and Laitinen, 1997). To succeed in this area Humphrey (2000) proposes that a CM plan be developed in conjunction with the establishment of a configuration control board to manage changes to all of the baseline configuration items and to ensure that configuration control procedures are followed.

A number of ‘best practice’ software process improvement (SPI) models such as ISO/IEC 15504 (also known as ‘SPICE’) and Capability Maturity Model Integration (CMMI) have been designed to help companies manage their software development activity. For example, CMMI is an SPI improvement model which specifies recommended practices in specific process areas – including CM - that have been shown to enhance software development and maintenance capability (Chriissis et al., 1991). This paper will investigate how thorough current medical device regulations are in relation to specifying what CM practices medical device companies should adopt when developing software. This will be achieved through comparing current medical device regulations and guidelines for CM against the formally documented software engineering ‘best practices’ of the CMMI for the CM process area.

2 MEDICAL DEVICE INDUSTRY

Medical device companies have to adhere to medical device regulations in relation to CM. Therefore the main area of concern for medical device companies in relation to CM is to ensure that the checklist of CM elements required by Food and Drug Administration (FDA) are in place rather than trying to improve their overall CM practices. GAMP (2001) details CM practices that medical device companies may adopt in order to comply with medical device regulations, however no documentation exists within the medical device domain in relation to how such practices could be improved by incorporating practices from formal software engineering SPI models for CM. However, if we investigate other regulated industries such as the automotive and space industries we realise that these domains are not content with satisfying regulatory standards, but have proactively developed SPI models specifically for their domain so that they may continuously improve the development of their information systems to achieve higher levels of safety, greater efficiency, and a faster time to market, whilst seamlessly satisfying regulatory quality requirements.

The major process improvement frameworks that currently exist, namely ISO/IEC 15504 and CMMI, do not address the regulatory requirements of either the medical device, automotive or space industries. Therefore, a new SPI model (Automotive SIG, 2005) was developed specifically for the automotive industry, this model was based upon ISO/IEC15504 (ISO, 2003) and is referred to as ‘Automotive SPICE’. Likewise, a new ISO/IEC15504 based SPI model was developed specifically for the space industry, this model is known as SPIcE for SPACE (Cass and Volcker, 2000). Both of these models contain reference and assessment information in relation to how companies may improve their configuration management practices within their domain.

This paper will not address the issue of developing an entire SPI model for the medical device industry (see McCaffery et al, 2004 for full discussion), but shall instead focus upon the individual process area of CM. This work addresses an opportunity to integrate the regulatory issues and SPI mechanisms to achieve improvements that are critical to the CM of software for medical devices.

3 CMMED DEVELOPMENT

The CMMED (Configuration Management model for the MEDical device software industry) was initiated by work that one of the authors performed whilst performing research for the Centre for Software Process Technologies at the University of Ulster, Northern Ireland. This work is now progressing with Lero – the Irish Software Engineering Research Centre. The initial research work was assisted by the involvement of a steering group with a pilot of 5 medical device companies and a notified standards body (all based in Northern Ireland). Each of the five companies expressed a desire to have access to a CM model that would incorporate software process improvement practices and could fulfil the relevant medical device regulatory requirements. However, this work is now being extended to include medical device companies in the Republic of Ireland.

The CMMED may be defined as a set of activities that if performed at a base level will satisfy the CM guidelines specified in the medical device standards. However, CMMED also enables medical device companies to follow a SPI path to achieving
CMMI certification. The CMMED will be flexible in that relevant elements of the model may be adopted as required to provide the most significant benefit to the business. The model is based on the CMMI, however another model is also being developed that is based upon ISO/IEC15504. The regulations used to extend the CMMI framework will be those of the FDA and the ANSI/AAMI SW68:2001 (SW68) standard (Medical device software – Software life cycle processes).

The CMMED will provide a means of assessing the software engineering capability for the configuration management process area in relation to software embedded in medical devices (FDA/CDRH, 1997, 1999, 2005). The CMMED is being developed to promote SPI practices into the CM process adopted by medical device companies. This is an attempt to improve the effectiveness and efficiency of CM within medical device companies through investigating the mapping of medical device regulatory guidelines against the CMMI CM process area.

The mappings between the medical device standards and the CMMI specific practices for the CM process result in the CMMED being composed of a number of goals, practices and activities. The CMMED determines what parts of the CMMI CM process area are required to satisfy medical device regulations. It also investigates the possibility of extending the CMMI process areas with additional practices that are outside the remit of CMMI, but are required in order to satisfy medical device regulatory guidelines.

The following section will detail a mapping of existing software development and regulatory guidelines for the medical device industry against the CMMI for the CM process area.

### 4 GUIDELINE MAPPING

The FDA provides little insight into how CM should be performed other than to state that a CM plan should exist and that this should be adopted to manage configuration items for medical device software. Therefore in order to gain a greater understanding of the CM guidelines that medical device companies follow in order to achieve regulatory compliance we referred to the medical device software life cycle processes (SW68) standard. This standard was drafted for use in the medical device sector based on the lifecycle requirements of ISO/IEC 12207 (ISO, 1995). This section illustrates the CMMED structure for the CM process area. In order to achieve this, FDA regulations & SW68 guidelines (for the rest of the paper we refer to these together as medical device standards) were mapped against the goals and practices of the CMMI CM process area.

This mapping is presented as follows: Firstly, we identify the goals that exist within the CMMI CM process area. Next the CMMI CM practices are identified within each CM goal. Then the CM activities (associated with the current practice) that have to be performed in order to comply with medical device regulations are listed. We then identify the activities that have to performed in order to adhere to the CMMI in relation to the current practice. Finally we lists the CMMI CM activities that are required in order to meet the medical device regulatory requirements associated with the current practice. The composition of the resulting CMMED is illustrated in figure 1.

![Figure 1: Composition of the CMMED.](image_url)

It should be noted however, in some instances the CMMI CM activities associated with the current practice may not provide full coverage of the medical device standards and therefore these additional activities have to be added in order to achieve the full list of activities required to fulfil the objectives of CMMED.

The CMMED has three goals: Goal 1: Establish Baselines, Goal 2: Track and Control Changes and Goal 3: Establish Integrity. To meet each of these goals it is necessary for a number of practices and activities to be performed. Each of the following sub-sections will present the CM activities required for each of the 3 goals.

#### 4.1 Goal 1: Establish Baselines

In order to fulfil Goal 1 Establish Baselines the following practices have to be performed: Identify Configuration Items, Establish a CM System and Create or Release Baselines.
4.1.1 Identify Configuration Items

The 4 activities that have to be performed in order to achieve regulatory compliance in relation to identifying configuration items are:
1. Select the configuration items and the work products that compose them based on documented criteria
2. Assign unique identifiers to configuration items
3. Specify when each configuration item is placed under CM
4. Identify Off the Shelf Components

The 5 activities that have to be performed in order to satisfy the CMMI practice for identifying configuration items are:
1. Select the configuration items and the work products that compose them based on documented criteria
2. Assign unique identifiers to configuration items
3. Specify the important characteristics of each configuration
4. Specify when each configuration item is placed under CM
5. Identify the owner responsible for each configuration item

The 3 activities that are common to both the CMMI and the medical device standards for identifying configuration items are:
1. Select the configuration items and the work products that compose them based on documented criteria
2. Assign unique identifiers to configuration items
3. Specify when each item is placed under CM

Therefore, in order to adhere to the medical device standards only 3 out of the 5 activities required for the CMMI in relation to identifying configuration items are necessary. However an additional activity is required in order to identify Off-the-Shelf (OTS) components as this is not included in the CMMI. Therefore 4 CMMED activities are required for identifying configuration items are:
1. Select the configuration items and the work products that compose them based on documented criteria
2. Assign unique identifiers to configuration items
3. Specify when each configuration item is placed under CM
4. Identify Off the Shelf Components. Note: this activity is not present in the CMMI but is required in order to fulfil the requirements specified in the medical device standards.

4.1.2 Establish a CM System

The 2 activities that have to be performed in order to achieve regulatory compliance in relation to establishing a configuration management system (CMS) are:
1. Store and retrieve configuration items in the CM system
2. Store, update, and retrieve CM records

The 8 sub-practices that have to be performed in order to satisfy the CMMI practice for establishing a CMS are:
1. Establish a mechanism to manage multiple control levels of CM
2. Store/retrieve configuration items in the CMS
3. Share and transfer configuration items between control levels within the CMS
4. Store and recover archived versions of configuration items
5. Store, update, and retrieve CM records
6. Create CM reports from the CMS
7. Preserve the contents of the CMS
8. Revise the CM structure as necessary

There are 2 activities that are common to both the CMMI and the medical device standards for establishing a CMS. Therefore, in order to adhere to the medical device standards, only 2 of the 8 activities required by the CMMI for establishing a CMS are necessary. The main differences are that CMMI requests the usage of multiple control levels of CM, as well as archiving and restoration procedures to be in place. The 2 CMMED activities for establishing a CMS are:
1. Store and retrieve configuration items in the CM system
2. Store, update, and retrieve CM records

4.1.3 Create or Release Baseline

There is only a single activity that has to be performed in order to adhere to the medical device standards in relation to creating or releasing baselines - Document the set of configuration items that are contained in a baseline. Whereas there are 4 activities that have to be performed in order to satisfy the CMMI practice for creating or releasing baselines:
1. Obtain authorisation from the CCB before creating or releasing baselines of configuration items
2. Create or release baselines only from configuration items in the CM system
3. Document the set of configuration items that are contained in a baseline
4. Make the current set of baselines readily available

There is only single CMMED activity that is common to both the CMMI and medical device standards for creating or releasing baselines. Therefore, in order to adhere to the medical device standards only one of the 4 activities - Document the set of configuration items that are contained in a baseline – is required for the associated CMMI practice is necessary.

4.1.4 Summary of CMMED Goal 1

Table 1 summarises goal 1 of CMMED (Establish Baselines). It may be observed from table 1 that not all of activities of the CMMI have to be performed in order to satisfy the medical device regulations (in fact only 6 of the 17 CMMI activities have to be performed). However, in order to satisfy the objectives of the CMMED 1 additional (medical device specific) activity had to be added (i.e. to satisfy goal 1 of the CMMED).

Table 1: Summary of CMMED Goal 1.

<table>
<thead>
<tr>
<th>Practice</th>
<th>CMMI activities</th>
<th>CMMI activities to meet medical device standards</th>
<th>Additional activities to meet medical device standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify CM items</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Establish a CMS</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Create or delete Baselines</td>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>17</strong></td>
<td><strong>6</strong></td>
<td><strong>1</strong></td>
</tr>
</tbody>
</table>

4.2 Goal 2: Track and Control Changes

In order to adhere to the CMMED goal 2 of tracking and controlling changes, the following specific practices have to be performed: Track Change Requests and Control Configuration Items.

4.2.1 Track Change Requests

The 5 activities that have to be performed in order to achieve regulatory compliance in relation to tracking change requests:
1. Initiate and record change requests in the change request database
2. Analyse the impact of changes and fixes proposed in the change requests.
3. Review change requests that will be addressed in the next baseline with those who will be affected by the changes and get their agreement.
4. Track the status of change requests to closure.
5. Each upgrade, bug fix, or patch for OTS software shall be evaluated, and the evaluation shall be documented.

There are 4 activities that have to be performed in order to satisfy the CMMI practice for tracking change requests:
1. Initiate and record change requests in the change request database
2. Analyse the impact of changes and fixes proposed in the change requests.
3. Review change requests that will be addressed in the next baseline with those who will be affected by the changes and get their agreement.
4. Track the status of change requests to closure.

There are 4 activities that are common to both the CMMI and the medical device standards for tracking change requests:
1. Initiate and record change requests in the change request database
2. Analyse the impact of changes and fixes proposed in the change requests.
3. Review change requests that will be addressed in the next baseline with those who will be affected by the changes and get their agreement.
4. Track the status of change requests to closure.

Therefore, in order to adhere to the medical device standards all of the activities required for this CMMI practice are necessary, but not always to the same level of detail. However an additional practice is required in order to ensure that each upgrade, bug fix, or patch for OTS software is identified and evaluated, and that the evaluation is documented, as this is not included in the associated CMMI practice.

The CMMED activities for tracking change requests are:
1. Initiate and record change requests in the change request database
2. Analyse the impact of changes and fixes proposed in the change requests.
3. Review change requests that will be addressed in the next baseline with those who will be affected by the changes and get their agreement.
4. Track the status of change requests to closure.
5. Each upgrade, bug fix, or patch for OTS software shall be evaluated, and the evaluation shall be documented. **Note: this activity is not present in the CMMI but is required in order to**
fulfil the requirements specified in the medical device standards.

4.2.2 Control Configuration Items

The 4 activities that have to be performed in order to achieve regulatory compliance in relation to controlling configuration items are:

1. Control changes to configuration items throughout the life of the product
2. Obtain appropriate authorisation before changed configuration items are entered into the CM system
3. Perform reviews to ensure that changes have not caused unintended effects on the baselines
4. Record changes to configuration items and the reasons for the changes as appropriate

The 5 activities that have to be performed in order to satisfy the CMMI practice to control configuration items are:

1. Control changes to configuration items throughout the life of the product
2. Obtain appropriate authorisation before changed configuration items are entered into the CM system
3. Check in and check out configuration items from the CM system for incorporation of changes in a manner that maintains the correctness and integrity of the configuration items
4. Perform reviews to ensure that changes have not caused unintended effects on the baselines
5. Record changes to configuration items and the reasons for the changes as appropriate

As the control of configuration items is very important in terms of ensuring the integrity of medical device software it is no surprise that 4 of the 5 activities required for this CMMI practice are necessary in order to adhere to the medical device standards.

The following list shows the mapping of the medical device standards against each of the activities required by the CMMI practice for controlling configuration items:

<table>
<thead>
<tr>
<th>Practice</th>
<th>CMMI activities</th>
<th>CMMI activities to meet medical device standards</th>
<th>Additional activities to meet medical device standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track change requests</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Control Config items</td>
<td>5</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>9</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

4.2.3 Summary of CMMED Goal 2

Table 2, summarises goal 2 of the CMMED (Track and Control Changes). It may be observed that almost all of the activities of this CMMI goal will have to be performed in order to satisfy the medical device standards (in fact 8 of the 9 CMMI sub-practices will have to be performed). However, in order to satisfy the objectives of CMMED 1 additional sub-practice had to be added.

4.3 Goal 3: Establish Integrity

In order to fulfil CMMED goal 3: Establish Integrity the following specific practices have to be performed: Establish CM Records and Perform Configuration Audits.

4.3.1 Establish CM Records

The 3 activities that have to be performed in order to achieve regulatory compliance in relation to establishing CM records are:

1. Record CM actions in sufficient detail so the content and status of each configuration item is known and previous versions can be recovered
2. Identify the version of the configuration items that constitute a particular baseline.
3. Revise the status and history of the configuration item as necessary

The 6 activities that have to be performed in order to satisfy the CMMI practice for establishing CM records are:

1. Record CM actions in sufficient detail so the content and status of each configuration item is known and previous versions can be recovered
2. Ensure that relevant stakeholders have access to and knowledge of the configuration items
3. Specify the latest version of the baseline.
4. Identify the version of the configuration items that constitute a particular baseline.
5. Describe the differences between successive baselines.
6. Revise the status and history of the configuration item as necessary.

The process of establishing CM records is very important in terms of providing the traceability evidence that is required to meet the regulatory requirements associated with medical device software. Half of the activities (3 out of 6) required for this CMMI practice are necessary in order to adhere to the medical device standards and are therefore included in CMMED. The CMMED activities for establishing CM records are:
1. Record CM actions in sufficient detail so the content and status of each configuration item is known and previous versions can be recovered
2. Identify the version of the configuration items that constitute a particular baseline.
3. Revise the status and history of the configuration item as necessary.

4.3.2 Perform Configuration Audits

The medical device standards do not specify any activities that have to be performed in order to achieve regulatory compliance in relation to performing configuration audits. The list of the sub-activities that have to be performed in order to satisfy the CMMI practice for performing configuration audits are:
1. Assess the integrity of the baselines
2. Confirm configuration records correctly identify the configuration of the configuration items
3. Review the structure and integrity of the items in the CM system
4. Confirm the completeness and correctness of the items in the CM system
5. Confirm compliance with applicable CM standards and procedures
6. Track action items from the audit to closure

This practice in CMMI has no equivalent practice within the medical device regulations. The medical device regulations do not specify any need for auditing the CM processes and activities. Therefore CMMED contains no activities, as a result of mapping the regulatory medical device requirements for CM against each of the activities required for the CMMI practice relating to performing configuration audits.

4.3.3 Summary of CMMED Goal 3

Table 3 summaries goal 3 of the CMMED (Establish Integrity). It may now be determined that in order to satisfy medical device standards that not all of activities of this CMMI goal have to be performed (in fact only 3 of the 12 CMMI activities have to be performed. Additionally, no additional (medical device specific) activities have to be added in order to satisfy the objectives of CMMED.

<table>
<thead>
<tr>
<th>Practice</th>
<th>CMMI activities</th>
<th>CMMI activities to meet medical device standards</th>
<th>Additional activities to meet medical device standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establish CM records</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Perform configuration audits</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>12</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

5 PRELIMINARY FEEDBACK

In order to assist with preliminary feedback, the CM process outlined by this paper has been compared against the existing practices within an Irish medical device company. A high level summary of their comments are included below.

They liked the structure of the CMMED and in particular how it enabled them to create a list of all the CM practices that they should adopt in order to adhere to the medical device standards. They also made positive comments in relation to CMMED providing additional information in relation to how their existing CM practices could be improved by incorporating guidance from the CM CMMI process area in relation how mandatory medical device activities may be performed.

Upon further consultation with the authors it has also been decided that in order to assist with SPI within the company that a process diagram shall be created, this will provide a graphical representation of the logical flow of the practices within their CM process.
6 SUMMARY AND CONCLUSION

Table 4 provides a summary of the 3 goals within CMMED. There are 40 activities required by CMMED, consisting of 38 CMMI and 2 medical device specific activities. In order to satisfy the mandatory medical device CM requirements, 19 of these activities have to be adhered to (17 CMMI and 2 medical device specific activities).

Table 4: Summary of CMMED Goals.

<table>
<thead>
<tr>
<th>CMMED goal</th>
<th>CMMI activities</th>
<th>CMMI activities to meet medical device requirements</th>
<th>Additional activities to meet medical device requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal 1</td>
<td>17</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Goal 2</td>
<td>9</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Goal 3</td>
<td>12</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>38</td>
<td>17</td>
<td>2</td>
</tr>
</tbody>
</table>

It is clear that following the guidelines specified in the medical device regulations will at best, only partially meet the specific goals of this CMMI process area (this would only fulfill 17 of the 38 activities required by CMMI). Since failure to perform any specific practice implies failure to meet the specific goal, with respect to CMMI, it is clear, that the goals of CM cannot be obtained by satisfying medical device regulations and guidelines during software development. But is the opposite true, can meeting the CMMI goals for CM successfully meet FDA and SW68 guidelines? With the exception of 2 sub-practices, performing the CMMI specific practices for CM would in general more than meet the FDA and SW68 guidelines for this area.

If a medical device company follows the CMMI guidelines for CM (with the exception of 2 activities), this will more than fulfill the CM requirements specified in the medical device regulations. However, only a fraction of the CMMI guidelines for CM will be satisfied by adhering to the medical device regulations for CM.

ACKNOWLEDGEMENTS

This research is supported by the Science Foundation Ireland (SFI) funded project, Global Software Development in Small to Medium Sized Enterprises as part of Lero - the Irish Software Engineering Research Centre (http://www.lero.ie).

REFERENCES

FDA/CDRH Guidance Document, Guidance for Off-the-Shelf Software Use in Medical Devices, FDA, September 1999
Medical device software life cycle processes, American National Standard / Association for the Advancement of Medical Instrumentation, SW68, 2001.
BRIDGING BETWEEN MIDDLEWARE SYSTEMS: OPTIMISATIONS USING DOWNLOADABLE CODE

Jan Newmarch
Faculty of Information Technology, Monash University
Melbourne, Australia
jan.newmarch@infotech.monash.edu.au

Keywords: Middleware, UPnP, Jini, Service oriented architecture, downloadable code, proxies.

Abstract: There are multiple middleware systems and no single system is likely to become predominant. There is therefore an interoperability requirement between clients and services belonging to different middleware systems. Typically this is done by a bridge between invocation and discovery protocols. In this paper we introduce three design patterns based on a bridging service cache manager and dynamic proxies. This is illustrated by examples including a new custom lookup service which allows Jini clients to discover and invoke UPnP services. There is a detailed discussion of the pros and cons of each pattern.

1 INTRODUCTION

There are many middleware systems which often overlap in application domains. For example, UPnP is designed for devices in zero-configuration environments such as homes (UPnP Consortium, 2006). Jini is designed for adhoc environments with the capability of handling short as well as long-lived services (Waldo, 2005) while Web Services are designed for long running services across the Web (WWW Consortium, 2002). There are many other middleware systems such as CORBA, Salutation, HAVi et each with their own preferred application space, and these different application spaces will generally overlap to some extent\(^1\).

It is unlikely that any single middleware will become predominant, so that the situation will arise where multiple services and clients exist but belonging to different middleware systems. To avoid middleware “silos”, it is important to examine ways in which clients using one middleware framework can communicate with services using another.

This issue is not new: the standard approach is to build a “bridge” which is a two-sided component that uses one middleware on one side and another middleware on the other. Examples include Jini to CORBA (Newmarch, 2001), Jini to UPnP (Allard et al., 2003), SLP to UPnP, etc. These essentially replace an end-to-end communication between client and service by an end-to-middle-to-end communication, where the middle (the bridge) performs translation from one protocol to the other.

Newmarch (Newmarch, 2005) has investigated how a Jini lookup service can be embedded into a UPnP device to provide an alternative to the bridging architecture. However, in practical terms this is an invasive mechanism which requires changes to the UPnP device and cannot be easily retro-fitted into devices.

Jini (Arnold, 2001) is apparently unique in production-quality middleware systems with service discovery in that rather than giving some sort of remote reference to clients it downloads a proxy object into the client (the proxy is a Java object). Many of the obvious security issues in this have already been addressed by Jini. It has also been claimed that this will lead “to the end of protocols” (Waldo, 2000). In this paper we investigate the implications of downloadable code for bridging systems, and show that it can lead to many optimisations.

Some of our work can be applied to middleware systems which support downloadable code but not discovery, such as JavaScript in HTML pages.

We illustrate some of these optimisations with a Jini-to-Web Services bridge and others with Jini-to-UPnP bridge.

\(^1\)The middleware systems we are interested in involve discovery of services, rather than just transport-level middleware such as HTTP connecting web browsers and HTTP servers.
The principal contribution of this paper is that it proposes and demonstrates a number of optimisations that could be considered to be additional architectural patterns that can sometimes be applied to bridge between different middleware systems. The validity of these patterns are demonstrated by discussion of several example systems and through an implementation for bridging between UPnP services and Jini clients. However, the patterns do have strong requirements on the client-side middleware: it must be possible to dynamically download code to clients and to dynamically determine the content of this downloaded code.

The structure of this paper is as follows: the next section discusses some general properties of bridging systems and the following section discusses downloadable code in this context. Section 4 introduces the first of three optimisations, one for transport-level bridging. Section 5 considers service cache management and the following section applies this to the second optimisation, for service-level transport. This is followed by a section on device-level optimisation. Successive sections deal with event handling and the implementation of a Jini-UPnP bridge based on these principles. We then assess the proposals and consider the value and generality of our work, before a concluding section.

Background knowledge of Jini may be found in Newmarch (Newmarch, 2001) and on the UPnP home site (UPnP Consortium, 2006).

2 BRIDGING

Nakazawa et al (Nakazawa et al., 2006) discuss general properties of middleware bridges. They distinguish three features

- Transport-level bridging concerns translation between two invocation protocols where a client makes a request of a service. Examples of invocation protocols include SOAP and CORBA’s IIOP. Transport-level bridging is concerned with translating from the invocation of a request to its delivery, and also between any replies.

- Service-level bridging involves the advertisement and discovery of services. Examples of discovery protocol include CORBA’s use of a Naming service and UPnP’s Simple Service Discovery Protocol.

- Device-level bridging concerns the semantics of services.

Transport level bridging includes translating between the data-types carried by each protocol. For example for Web Services using SOAP, these are XML data-types while for Java RMI using JRMP these are serialisable Java objects. There are usually problems involved in such conversions. Vinoski (Vinoski, 2005) points to the mismatch between Java data-types and XML data-types. While he goes on to examine the consequences for JAX-RPC, the same issues cause problems converting from SOAP data-types to Java objects on JRMP. Newmarch (Newmarch, 2005) discusses the mismatch between UPnP data-types and Java objects and concludes that the UPnP to Java mapping is generally okay but the opposite direction is not. There is no general solution to the data-mapping problem, and indeed the use of the so-called “language independent” XML in some middleware systems appears to have exacerbated this. Services where the data-types are not convertable cannot be bridged. This paper does not address this issue.

While the transport protocol is usually end-to-end, the discovery protocol may be either end-to-end as in UPnP or involve a third party. Dabrowski and Mills (Dabrowski and Mills, 2001) term this third-party a service cache manager (SCM). Examples of such a manager are the Jini lookup service, the CORBA and RMI Naming service and UDDI (although this does not seem to be heavily used). The implications for service-level bridging involve the discovery protocol: in an end-to-end discovery system the service-level bridge will need to understand how to talk directly to services and/or clients, while with a service cache manager the bridge will need to understand how to talk to the service cache manager.

Device-level bridging concerns the meaning of “service” in different middleware systems, and how services (and devices) are represented.

In general a bridge system will look like Figure 1.

3 DOWNLOADABLE CODE

There are many examples where code is downloaded from one computer to execute in another. These include JavaScript in HTML pages, Safe-Tcl (Levy et al., 1997) and Erlang (Brown and Sablin, 1999). Jini as a service-oriented architecture makes use of RMI to download a proxy object representing a service into a client. This changes the nature of the client/service transport protocol since that is now managed by the proxy object, not by the client-the
The Java Extensible Remote Invocation framework (Jeri) in Jini 2.0 allows the proxy and service to use any protocol that they choose.

Proxy/service communication in Jini can be represented in Figure 2.

Figure 2: Proxy communication.

The pattern of communication of Figure 2 can also be employed by JavaScript using the Ajax extensions (Garrett, 2005), and is used by Google Maps and Google Mail for example, although the communication is restricted to HTTP calls.

4 OPTIMISING TRANSPORT-LEVEL BRIDGING

Transport-level bridging involves the bridge receiving messages from a client using the client’s transport protocol, translating them into messages for the service and sending them using the service’s transport protocol. Responses are handled in a similar way.

Many internet protocols specify all components of the interaction between clients, services and service cache managers. For example, UPnP specifies the search and discovery protocols and also the protocol for procedure call interaction between client and service as SOAP. However, as was shown by Java RMI over CORBA’s IIOP instead of JRMP, and also by CORBA’s use of Naming and Trader services, there is no necessary link between discovery and invocation. As long as a client and service are using the same invocation protocol they can interact directly.

For UPnP and many systems there is little choice since the invocation protocol is fixed by the middleware specification. However, Jini 2.0 allows a “plug-gable” communications protocol. While most systems would require the client to have the communications protocol “hard coded” (or loadable from local files), Jini allows a service proxy to be downloaded from a lookup service (service cache manager) to a client, and this can carry code to implement any desired communication protocol.

In a similar but less flexible way, the Ajax XMLHttpRequest object can exchange any type of data with its originating service. Usually this is XML data, but could be other types such as JSON (JSON, 2006)

In the most common situations, the service proxy communicates with its bridge service. However, a transport-level bridge is just there to translate and communicate between the client and the service. If the code to do this translation is moved into the proxy, then the transport-level component of the bridge service becomes redundant. That is, the client makes local calls on the proxy, which makes calls directly to the service using the service’s transport protocol. One leg of the middleware has been removed. This is illustrated in Figure 3.

Figure 3: Removing one transport step.

This optimisation improves performance by
- removing one serialisation step
- removing one deserialisation step
- removing one network transport leg

In addition, the conversion to the destination protocol is performed once at the client-side. There are some systems such as that of Nakazawa et al (Nakazawa et al., 2006) in which the bridge performs conversion from source data-types to an intermediate “standard” type and from there to the destination type. This (or even just conversion from source transport data-types to destination transport data-types) introduce possibilities for semantic problems which are mitigated by a single conversion step at the client-side.

This pattern has been used by Newmarch (Newmarch, 2006) to show how a Jini client can communicate with a Web Service. The proxy uses SOAP, the transport protocol for the Web Service. The conversion from Java data-types to XML data-types is performed by the JAX-RPC package (which cannot do a perfect conversion job, as mentioned earlier). The role of the bridge is just there to advertise the Web Service to the Jini federation and to upload a proxy to the Jini lookup service.

This pattern can also be used by Jini clients to talk to CORBA services, since Jini can directly generate proxies that use IIOP.

Casati (Casati, 2006) shows how JavaScript downloaded into a browser can talk directly to Web Services instead of the more usual HTML-Servlet-Web Service (or similar) bridge (as typified by
the web site www.xmethods.com. Casati employs the XMLHttpRequest object which allows a browser to communicate with an HTTP server asynchronously. This is usually used to exchange data between the browser and original page server. But as Web Services typically use SOAP over HTTP, Casati gives JavaScript for the object to be used as a proxy to talk directly to the Web Service.

In a later section we discuss how we use this pattern for a Jini client to talk to a UPnP service.

5 SERVICE CACHE MANAGER

Service cache managers are expected to store “services” in some format and deliver them to clients. The stored service can be a simple name/address pair as in naming systems such as Java RMI or CORBA, complex XML structures linked to WSDL URLs for Web Services in UDDI directories, or other possibilities. The Jini lookup service stores service proxy objects, along with type information to locate them.

When clients and services are trying to locate a service cache manager, there is often an assumed symmetry, that the client and service are searching for the same thing. In our examples above, this occurs in all of naming services, UDDI registries and Jini lookup services.

Once found though, clients and service do different things: services register whereas clients look for services. The Jini ServiceRegistrar for example contains two sets of methods, one for services (register()) and one for clients (lookup()). UDDI similarly has two sets of messages, but there are more of them since UDDI has a more complex structure (Bellwood, 2002). Conceptually, there should be one protocol for services discovering caches and another for clients discovering them, with different interfaces exposed to each.

6 OPTIMISING SERVICE-LEVEL BRIDGING

The standard bridge acts as a client to one discovery protocol and as service to the other. For example, in a Jini/UPnP bridge (Allard et al., 2003) UPnP device advertisements are heard by a bridge acting as a UPnP control point, which re-advertises the service as a Jini service. In addition, it also acts as a transport-level bridge.

As a second optimisation we propose folding the service cache managers into the bridge, to just leave service-level bridging as in Figure 4.

As an illustration of this, we have built a lookup service as a service-level bridge which listens for UPnP device advertisements on one side. It can handle device registration and device farewells and will deal with device renewals, timing out if they are not received. In this respect it acts like a UPnP control point, but unlike a control point it does not send any action calls to the UPnP device or register itself for events. The other side of the service-level bridge handles requests from Jini clients, primarily a discovery request for the lookup service.

The lookup service will act like a normal Jini lookup service as far as the Jini client is concerned and return a lookup service proxy. The Jini client will be a normal Jini client and uses the lookup service to search for a service using the standard Jini API. If the lookup service knows of UPnP devices that deliver the service, it will prepare a proxy for the UPnP device and send it back to the Jini client.

This optimisation is only useful in conjunction with the first one. Transport-level bridging or its replacement will still need to be in place. If there is no replacement then little is gained by separating transport-level and service-level bridging. However, when the transport-level bridge is replaced by a smart proxy then it is possible to just keep the service-level bridge.

This is at present a practical restriction on the applicability of this pattern, since there do not appear to be many middleware systems in practical use apart from Jini that support both downloadable proxies and discovery services. However, this could be expected to change with future development of more advanced service-oriented frameworks (for example, see Edwards (Edwards et al., 2005)).

7 DEVICE-LEVEL BRIDGING

Different middleware systems have different basic ideas of services. Many systems such as CORBA, Jini and WebServices only have the notion of services. Others like UPnP and Bluetooth have devices. UPnP devices contain a number of services (and possibly other devices, recursively).
The different systems give different meanings to discovery. For example, the UPnP on/off light is a BinaryLight device containing a SwitchPower service. Jini has no concept of BinaryLight's and can only look for a SwitchPower service. So a Jini client cannot search for a binary light device but only some subset (as a collection) of the service interfaces offered. On the other hand, UPnP advertises the binary light device and the services, but with separate messages for each service, rather than as a group. UPnP devices usually only have one service although some may have more. For example, an internet gateway device may have several services and embedded devices. This device has a total service list of Layer3Forwarding, WANCommonInterfaceConfig, WANDSSLLinkConfig and WANPPPConnection. In general, a Jini service may implement a number of service interfaces, and a Jini client may request a service that simultaneously implements a number of interfaces.

In the case of UPnP, services are described by XML documents, while Jini services are described by Java interfaces. We have defined a standard mapping from UPnP services to Jini services. For example, the UPnP service description for a SwitchPower service is

```xml
<scp xmlns="urn:schemas-upnp-org:..."> ...
  <actionList>
    <action>
      <name>SetTarget</name>
      <argumentList>
        <argument>
          <name>newTargetValue</name>
          <relatedStateVariable>Target</relatedStateVariable>
          <direction>in</direction>
        </argument>
      </argumentList>
    </action>
  ...
</actionList>
</scp>
```

Our mapping translates this into the Java interface (along with a suitable definition of Target)

```java
public interface SwitchPower extends Remote {
  void SetTarget(Target newTargetValue) throws RemoteException;
  ...
}
```

The service-level bridge will need to be able to translate from one representation to the other. A direct approach is to store a table mapping each service. In the case of UPnP and Jini, the table would just hold UPnP service names matched to Java class files. At this stage, the service-level bridge will be responsible for creating the proxy, and to do this it needs the class files for the service interfaces. While it would be fine for the bridge to have class files for the set of "standard" devices and services maintained by the UPnP Consortium, it would not allow for new, unknown services to be managed.

New UPnP services would require the service-transport bridge to examine in detail the UPnP service description and generate source code for the Java interface. Then compile this on the fly using a local Java compiler (such as javac or Kirby's dynamic compiler (Kirby, 2005)). This is similar to dynamic compilation of JSP and servlets by servlet engines such as Tomcat. The resultant class files can be cached against repeated use.

Similar mechanisms may be needed for bridging between any middleware systems where new and unknown services may be presented to the service-level bridge. This will depend on what information is required by the bridge in order to create a proxy.

### 8 Optimising Device-Level Bridging

In the architecture proposed so far, the service-level bridge needs to be able to generate a proxy to represent the original service. For a Jini client, this requires class files on the lookup service for the Java interfaces, and for unknown service types these will need to be generated by the bridge. This will involve detailed introspection of the service descriptions and use of a Java compiler. While dynamic compilation of JSP pages demonstrates that this is feasible, it nevertheless has overheads.

The Jini client on the other hand has to know the service interface, otherwise it cannot ask for a service proxy. So if knowledge of the Java interfaces can be deferred to the client side, then it just becomes a lookup of already instantiated classes. The name of the interface is all that is required for the client to find the interface class.

The Jini lookup service already downloads a proxy to the client to represent it. This has not been shown in the figures so far as it is a Jini-specific (but standard) detail. Usually this proxy just makes remote calls back to the lookup service. However, just like any downloaded code, the proxy can be designed to perform any functions on the client side (subject to security constraints). In particular, on a lookup operation the proxy could just pass back to the lookup service.

---

2The client has to know the interfaces it is interested in. It should not know the implementation classes. This is addressed in the implementation section.
vice enough to allow a match to be made, and on success the lookup service could pass back just enough for the lookup service’s proxy to create a proxy for the original service. In the case of a UPnP/Jini bridge, the minimal information is the names of the interfaces required, and the returned information just needs to be the URL of the UPnP device description. These are enough for a proxy to be created on the client-side that can talk to the UPnP service. See Figure 5 for the final system.

![Figure 5: Optimised service-level bridging.](image)

### 9 EVENT HANDLING

The discussion so far has used the remote procedure call paradigm. However, there are other possibilities such as an asynchronous callback mechanism where the service makes calls back to the client. This is easily handled by the proposed systems, as the proxy just registers itself as the callback address.

### 10 IMPLEMENTATION OF OPTIMISED JINI-UPNP BRIDGING

There is an open source implementation of UPnP devices and control points by CyberGarage (Konno, 2006). This is very closely modelled on the UPnP Device Architecture specification (UPnP Consortium, 2006a). It exposes an API to allow a client to create a ControlPoint which can listen for device announcements, to determine the services within the device and it has methods to prepare parameters and make action calls on UPnP services. It also supports getting device information such as friendly name and registering as listener for state variable change events.

We use this in our lookup service to monitor UPnP devices and keep track of the services that are available, as well as device information.

The CyberGarage API treats UPnP devices and services using a DOM-oriented model, unlike the SOAP-oriented manner of Jini. We use the UPnP to Java mapping discussed earlier to translate between the two representations.

In our implementation, we use the Java Proxy class to give a dynamic proxy. This proxy implements all of the services on a UPnP device that are requested by the client. The proxy is supplied with the device URL so that it can access the device description. This description contains the URLs for action calls, for registering listeners and for the presentation. The Jini proxy requires an invocation handler. We use the CyberGarage classes to build a generic handler to deal with SOAP calls to the device. The CyberGarage classes and this handler are downloaded from the bridge to the client. This avoids the need for the bridge to know the service interfaces at all and allows the client to only know the service interfaces.

The proxy implementation uses the CyberGarage library, but only for the control components of the CyberGarage ControlPoint. That is, it is used to prepare and make SOAP action calls and to register and listen for UPnP events. However, it does not listen for devices, since that is done by the bridging lookup service. When a method call is made on the service proxy it uses the control point to make a SOAP remote procedure call.

Our current implementation relies heavily on the CyberGarage library, but only on the control point code. The device advertisement code is not used. Only a part of the control point code is used by the bridging lookup service to monitor devices while another part is used by the service proxy to make action calls and listen for events. However, the CyberGarage code is tightly interwoven, and it was not possible to use only the relevant parts. The lookup service has to import almost all of the library, as does the service proxy. It should be possible to produce a lightweight version for each with only the required partial functionality.

### 11 ASSESSMENT

Any “optimisation” often has both positive and negative sides. We try to offer a balanced viewpoint on the advantages and disadvantages of our pattern

#### 11.1 Transport-level Optimisation

In transport-level optimisation, we place the code to perform service invocation directly in the proxy downloaded to the client. The principal advantages of this are

- performance improvement by removing one serialisation step
• performance improvement by removing one deserialization step
• performance improvement by removing one network transport leg
• reducing the risk of semantic mismatches between client and service data types by reducing the number of data conversion steps.

The major disadvantage is that code has to be downloaded to the client that is capable of talking directly to the service. This is generally downloaded from an HTTP server. Some examples follow

• Casati (Casati, 2006) gives JavaScript that can be downloaded to a web browser such as Firefox or IE that can make function calls on Web Services. This requires just 10kbytes of JavaScript source code. This relies on the extensive libraries and support within the browser for many of the library calls made.

• Newmarch (Newmarch, 2006) discusses a Jini proxy that can make function calls on Web Services. The particular implementation used there makes use of the Apache Axis objects Call, QName and Service. These classes and all the classes they depend on are substantial in size—over 900kbytes. There are clear redundancies in this: for example, there are many classes which deal with WSDL document processing, and this is not needed by the proxy.

• For the Jini UPnP proxy discussed here, the CyberGarage classes are used. These classes are 270kbytes in size. However, the jar file also contains the source code for the package. Removing these reduces the size to 160kbytes and a specialised version could be even smaller. CyberGarage also requires an XML parser to interpret SOAP responses. The default parser (Xerces) and associated XML API package are over 1Mbyte in size which is substantial for an HTTP download. The kXML package can be used instead, and this is a much more reasonable 20kbytes and there is even a light version of this. This gives a total of 180kbytes which is acceptable for any Jini client—the reference implementation of Sun’s lookup service takes 50kbytes just by itself.

The actual amount of code downloaded depends on the complexity of the proxy and the degree of support that already exists in the client. These three examples show variations from 10kbytes to nearly 1Mbyte.

11.2 Service-level Optimisation

The standard bridge requires up to two service cache managers, one for each discovery protocol. In addition, the bridge has to act as a client to discover the original service and as a service to advertise to the original client. Service-level optimisation reduces this to two halves of two SCMs: one half to listen to service adverts, the other half for the original client to discover the service. UPnP does not have an SCM and control points listen directly to service adverts, which reduces the savings somewhat.

On the downside, it is necessary to write parts of service cache managers. Although this is not inherently difficult, knowledge of how to do this and API support by middleware systems is not so widespread as for writing simple clients and services. Jini has the necessary classes, but there are no tutorials on how to write a lookup service. CyberGarage has support for control points, but this is tightly woven with the device code and so contains redundant code.

In addition, the need to possibly perform introspection on service descriptions, to generate appropriate client-side definitions, to generate and compile them are disadvantages.

11.3 Device-level Optimisation

This optimisation gains by removal of some code (introspection, generation of interfaces and compilation) completely. On the other hand, code to generate the proxy is just moved into the client. In the case of Jini, most of this code is already present in the client from the Jini libraries and does not represent much of an overhead. For other systems it may be more costly.

11.4 Generality

The design patterns discussed in this paper rely on a number of properties of the two middleware systems in order to be applicable

• it must be possible for a service cache manager to be used in each middleware system. In practise this is not an onerous provision and it can be applied even to systems such as UPnP which do not require an SCM.

• There must be a (sufficiently good) mapping of the datatypes from service system to client system. This allows UPnP services to be called from Jini clients, but would limit the scope of Jini services that could be invoked by UPnP clients. As another example, the flexibility of XML data types means that it should be possible to mix Jini clients with Web Services, and Jini services with Web Service clients.

• It must be possible to download code from the SCM to run in either the client or service. In our case study, we have downloaded code to the client that understands the service invocation protocol, but it would work equally well if code could
be downloaded to the service that understands the client invocation protocol. Without this, the recipient would already need to know how to deal with a foreign invocation protocol, which would largely defeat the value of the pattern.

The third point is the most difficult to realise in practice. Many languages support dynamic code execution: most interpreted languages have an equivalent of the eval() mechanism, through to dynamic linking mechanisms such as dynamic link libraries of compiled, relocatable code. However, the only major language supporting dynamic downloads of code across a network appears to be Java, and the principal middleware system using this is Jini. Given some level of dynamic support, adding network capabilities to this is not hard: the author wrote a few pages of code as proof of concept to wrap around the Unix C dlopen() call to download compiled code across the network into a C program.

12 VALUE OF WORK

The value of mixing different middleware systems can be seen by a simple example. Through UPnP, various devices such as hardware-based clocks and alarms can be managed. A stock exchange service may be available as a Web Service. A calendar and diary service may be implemented purely in software as a Jini service. Using the techniques described in this paper, a Jini client could access all of these. Acting on events from UPnP clocks to trigger actions from the Jini diary the client could query the Web Service stock exchange service and ring UPnP alarms if the value of the owner’s shares has collapsed.

In addition to extending the use of clients and services, there are also some side benefits:

• Jini has suffered by a lack of standards work for Jini devices and device services, with a corresponding lack of actual devices. This work allows Jini to “piggyback” on the work done now and in the future by the UPnP Consortium and to bring a range of standardised devices into the Jini environment. Jini clients will be able to invoke UPnP services in addition to services specifically designed for Jini.

• UPnP is a device-centric service architecture. It allows clients to use services on devices, but has no mechanism for UPnP clients to deal with software-only services since they cannot be readily expressed in UPnP. Work is ongoing within the UPnP Consortium to bring WSDL descriptions into the UPnP world. Jini clients on the other hand are agnostic to any hardware or software base, and can mix services of any type.

Both middleware systems have limitations—in the case of Jini, in the types of services that can be accessed, and in the case of UPnP, in the range of services that can be offered. Other middleware systems will have similar limitations. For example, Web Services tend to deal with long-lived services at well-known addresses whereas Jini can handle transient services.

13 CONCLUSION

We have proposed a set of alternative architectures to bridge between different middleware systems which uses a service cache bridge and a downloadable proxy understanding the service or client invocation protocol. In addition, we have used this between Jini and UPnP and we have automated the generation and runtime behaviour of this proxy from a UPnP specification. This has been demonstrated to give a simple solution for UPnP services and Jini clients. The techniques are applicable to any client protocol which supports downloadable code and any service protocol.

REFERENCES


ENGINEERING A COMPONENT LANGUAGE: COMPJAVA

Hans Albrecht Schmid, Marco Pfeifer
University of Applied Sciences Konstanz, Brauneggerstr. 55, D - 78462 Konstanz, Germany
schmidha@htwg-konstanz.de, mpfeifer@htwg-konstanz.de

Keywords: Components, Component language, Component composition, Component fragment, Connections.

Abstract: After first great enthusiasm about the new generation of component languages like ArchJava, ComponentJ and ACOEL, a closer inspection and use of these languages identified together with their strong points some smaller, but disturbing drawbacks. These might impede a wider acceptance of component languages, which would be harmful since the integration of architecture description with a programming language increases the emphasis on, and consequently the quality of application architecture. Therefore, we took an engineering approach to the construction of a new Java-based component language without these drawbacks. That means, we derived general component language requirements: designed a first language version meeting the requirements and developed a compiler; used it in several projects; and re-iterated three times through the same cycle with improved language versions. The result, called CompJava, which should be fairly stable by now, is presented in the paper.

1 INTRODUCTION

The new generation of component languages, like ArchJava (Aldrich, May 2002) (Aldrich, 2002), ComponentJ (Seco, 2000), ACOEL (Sreedhar, 2002), and to a smaller degree, KOALA (van Ommering, 2000) (van Ommering, 2002) made enthusiastic about the new way of program construction without reference handling. These languages integrate architecture description with a programming language. Thus, they push the more abstract architecture-description-language (ADL) based approach (see ADL classification framework (Medvidovic, 2000), (Medvidovic, 1999)) forward towards a direct use. Our experience confirms that this increases the emphasis on, and consequently the quality of application architecture.

However, a closer inspection and use of component languages identified together with their strong points some small, but disturbing drawbacks.

For example, ArchJava components behave like classes with regard to some aspects. A component class generates implicitly its type, and inheritance is defined primarily as implementation inheritance among components. Further, though a component is not a class, it may inherit implementation from a class. ArchJava re-defines constructs for concepts, like interfaces, which it shares with Java. ACOEL shows no symmetry with regard to the attachment of code to provided and required interfaces. More drawbacks and details are given in section 2.

It seems that these drawbacks might impede a larger acceptance and broader use of component languages. Therefore, we designed a new component language that does not have these drawbacks, following a sound engineering approach. We derived a list of component language requirements from the identified drawbacks. We constructed a component language that covers the requirements (the first version being available fall 2003). Then, we used the language in projects, and had three iterations with improved language definitions. Now, the language will be quite stable.

Section 3 gives an overview about distinguishing structuring principles of CompJava, and section 4 introduces its type concept. Section 5 shows how components are composed in a structured way from component fragments, and section 6 shows how they are composed from subcomponents. Section 7 presents dynamic architectures using a Web server example.

2 LANGUAGE REQUIREMENTS

This section describes drawbacks identified in component languages and derives specific requirements from them. These component language requirements complement general, but unlisted requirements, defined by a kind of intersection of the features of existing languages.
Embedded OO-Programming Language

A component language embeds a programming language and uses its constructs to implement components. ArchJava which embeds Java has ports with both provided and required interfaces. It defines the interfaces of a port either by listing, after the keyword provides or requires, operation specifications, or by listing method implementations. But you cannot define the interfaces of a port using Java interfaces. Thus, the identical concept “interface” is described by different constructs in the component language and the OO-language, which is certainly a drawback.

On the other hand, ArchJava allows to derive components from classes, like the worker component from the class Thread (Aldrich, May 2002). But how can a component, which is not a class, but a first-class citizen of its own, inherit implementation from a class?

Therefore, Requirement 1 is: a component language should not reinvent constructs for concepts it shares with its programming language. On the other hand, it should not intermingle differing concepts in the component language and programming language.

Component Inheritance

ArchJava transfers the type concept of class-based languages directly to components. It defines a component type implicitly as the type that is generated by a component class, and it defines inheritance in such a way that a derived component inherits from a base component both the component type and its implementation.

This has two drawbacks. A definition of a component type that is independent from the implementation is required to define e.g. a product line architecture or a component framework. A product line architecture defines product component types which are implemented by different product components. Similarly, a component framework defines a set of collaborating component types which are implemented by different product components. Therefore, Requirement 2 is: a component language should not reinvent constructs for concepts it shares with its programming language.

Component Encapsulation

ArchJava allows that a parent component invokes directly internal methods of a subcomponent which are not defined by a provided port. This breaks the encapsulation of the subcomponent. Further, a graceful evolution is inhibited since it is not possible that a sibling subcomponent invokes these methods instead of the parent component at a later point of the evolution. On the other hand, ACOEL allows that a parent component exposes a reference to a subcomponent in a port. When it passes that component reference to a sibling component, ports of the sibling component may be connected to ports of the subcomponent. That means a component may be at the same time a subcomponent of two different components. This breaks a sound architectural structure.

Requirement 3 is that a component should be completely encapsulated, i.e. it should collaborate only via its ports with external code. As a consequence, a subcomponent of a component must not collaborate with other components outside of its parent component. Therefore, the passing of component or port references should be restricted or prohibited.

Interface Symmetry

ArchJava has a complete symmetry among provided and required interfaces with regard to their definition and their use, since a port may comprise both of them. ACOEL (Sreedhar, 2002) has a symmetry with regard to their definition, but not with regard to their use. A mix-in allows to put a filter between a provided port and the implementing class. But it does not allow to put a filter between the implementing class and a required port.

Requirement 4 is that the definition and the handling of provided and required ports should be symmetrical.

Ports and Connectors

An ArchJava port may combine a provided and a required interface, like:

```
port port1 provides m1, m2 requires m3, m4;
```

As usual, a port with a required interface Iₙ may be connected to a port with a provided interface Iₙ when Iₙ is a subtype of Iₙ. But an ArchJava connector may fork the calls from a required interface Iₙ to several provided interfaces like Iₙ and Iₙ if each is a supertype of Iₙ, and their union is a subtype of Iₙ, and their intersection with regard to Iₙ is empty. For example, with port2 and port3:

```
port port2 provides m3, m6 requires m1, m5;
port port3 provides m4, m5, m6 requires m2, m3;
```

ArchJava allows to connect port1, port2, and port3 by a connect statement. If port1 would require additionally m6 the connection would not be correct and rejected. This is not easy to check and
understand for a programmer; it might be considered as a new kind of spaghetti problem (without dining philosophers). Though it is easy for a compiler to check what happens, we should disallow it.

**Requirement 5** is that the definition of ports and connectors should be made in a way that is easily understandable to a programmer.

**Collaboration of Subcomponent Ports with Code**
ArchJava defines private ports in order to connect component code with a port of a subcomponent. However, a private port is a contradiction in itself since the ports of a component define its interfaces to the outside, i.e. the points of collaboration with external code: So what is the semantics of a private port? It is even more confusing that ArchJava allows to connect two private ports; what does that mean? Our conclusion is that the concept of private ports is questionable. **Requirement 6** is that an adequate construct should connect component code with a port of a subcomponent.

**Implementation Isomorphy with OO-Based Approach**
ArchJava generates one component class which lists the provided methods of all public and private ports of the component. The generated code does not group together the methods which implement the operations of the same port. Similarly, the required operation of all ports are always invoked from that list of methods. There is no way to group the methods that invoke the operations of the same port. This is in contrast to the usual OO-based implementation of a component where the provided methods of each port are implemented by a different class, and the required methods of each port are usually invoked by methods from different classes.

Therefore, **requirement 7** is that the code generated from a component should have at least some isomorphy with corresponding code written in class-based OO-languages.

**Implementation Efficiency**
The efficiency of the code generated from a component language may not be a primary concern when large architectural components with powerful operations are realized. But in many cases, the efficiency of a frequently performed operation invocation matters. Consider e.g. a scanner, used e.g. as a subcomponent of a compiler, which is certainly not a lightweight component. If it fetches the next character from a source file over a required interface with a getCharacter-operation (compare section 6), the efficiency of that frequently performed operation invocation has a strong influence on the scanner overall performance.

**Requirement 8** is that the code generated from a component language should have about the same efficiency for basic constructs, like e.g. operation invocation over connected ports, as an equivalent (but not tricky) class-based implementation. We state that requirement due to its importance for a wide acceptance of component languages, though we cannot cover it in this paper for space reasons.

### 3 COMPJAVA OVERVIEW

Distinguishing features of CompJava are, besides the definition of component types and component type inheritance, its structuring facilities for component construction. CompJava allows not only, like the new generation of component languages, to compose components from subcomponents in a structured way. It allows to compose them also in the same way from code building blocks, or from a combination of subcomponents and filters formed by code building blocks.

CompJava introduces plugs which are used mainly for connecting component fragments with subcomponent ports.

Ports of subcomponents are connected with the connect-statement to other ports or plugs. Component fragments are attached to the inside of the component ports or to plugs with an attach-statement. Thus, CompJava allows to compose

1. a low-level component from component fragments, as illustrated by Figure 1 a)
2. a high-level component from subcomponents, as illustrated by Figure 1 b)
3. a medium/high-level component from a combination of subcomponents and component fragments that are used as filters, as illustrated by Figure 1 c)

in a clear, clean and structured way.
For a graphical depiction of the composition of a component, we have enriched UML 2 component diagrams with component fragments and plugs (depicted by a diamond). A component fragment is represented according to the selected implementation as an anonymous class, an inner class or as a method block (depicted like an anonymous class without class head).

![Diagram of component composition](image)

Figure 1: Composition of a component from component fragments (a), from subcomponents (b), and from a combination of them (c).

The first version of the CompJava compiler has been available since winter 2003/2004, three more versions followed. The new version to be available in fall 2006 will be integrated in Eclipse. The CompJava Designer is a graphical design tool that allows to draw enriched CompJava component diagrams and to generate component code skeletons. It is an Eclipse-plugin and in prototype stage, to be available spring 2007.

The following sections introduce the CompJava language and shows that their constructs satisfy the requirements. We use a compiler as a running example. The compiler component is composed from a scanner, parser and other subcomponents.

## 4 COMPONENT TYPES

Let us consider first the scanner component. We define the provided interface of the scanner as a Java interface. It includes all scanner-related responsibilities, like setting the file name of the source file to be processed, and fetching the next token from it.

```java
interface ScannerIF {
    Token getNext();
    void setSource(String sourceName);
}
```

Since the `ScannerIF` interface includes all source file processing related responsibilities, the component type `ScannerType` is defined with a single provided port.

```java
component type Scanner1Type {
    port in provides ScannerIF;
}
```

A component type defines all interfaces of a component. That means components are completely encapsulated: all methods in a component, except for the main method, can be invoked from outside only via provided ports, and all methods can invoke an outside method only via required ports.

A port has either a provided, a required or an event interface. A port declaration gives the port name and after the corresponding keyword the associated interface. An event port is similar to a required port, but its operations must not have results, and several provided ports of event listeners may be connected to it. As we show in section 7, a component type may also define port arrays or port vectors.

A component type may extend another component type, like an interface may extend another one. It inherits all ports, and it may extend the interface of inherited provided ports or may add provided ports.

## 5 LOW-LEVEL COMPONENTS

This section shows how low-level components are composed from component fragments.

### Implementing Provided Ports

A component has a component type (indicated by the ofType-clause). It implements all the provided ports, and may invoke operations from the required ports specified by its component type. In the Scanner1 component (see Figure 2), an attach-statement attaches the inside of the provided port in to a component fragment, an anonymous class implementing the `ScannerIF` interface.
component Scanner1 ofType Scanner1Type {
  //port in provides ScannerIF;
  attach This.in to new ScannerIF {
    private File sourceFile;
    void setSource(String name) { //open sourceFile}
    char getChar() { //next char from sourceFile}
    Token getNext() {
      Token current = new Token();
      char c = getChar();
      while ( c != separator ) {
        current.append( c );
        c = getChar();
      }
      return current;
    }
  }
}

Figure 2: The Scanner1 component with port in providing the ScannerIF implemented by a anonymous class.

An attach-statement may be used to attach the inside of a provided port to a component fragment that implements an interface I. The condition is that I extends (including equals) the port interface; it is checked at compile time. A component fragment may be a Java construct: an instance of an anonymous class, as shown, or an instance of an inner class. The inside of a port is indicated by the keyword This, which stands for the component instance, followed by the port name. The declaration of inner and anonymous classes follows the Java standard; the only difference is that they are used inside of a component instead of a class.

When a component, like Scanner1, is quite small and not composed from other components, it might be a disadvantage that its implementation generates two object instances: one of the application-specific component fragment and another one of the component class. Therefore, CompJava allows also that a component fragment is formed by a method block. A method block is a sequence of methods that implement a given interface (see Figure 3). A method block is not a Java construct, but an analogon to a Java block, which is a sequence of statements. When different provided ports are each attached to a method block, there is the restriction that their interfaces must have an empty intersection.

Consequently, CompJava provides component fragments which include method blocks, inner classes or anonymous classes, in order to structure the implementation of a component.

Accessing Required Ports
The Scanner mixes up two different concerns, scanning the program character stream, and handling of the source file to be parsed. Similarly, the ScannerIF interface mixes up two different concerns, accessing the tokens which the scanner creates, and determining the source file to be parsed. We should separate the different concerns, scanning and source file handling. To this purpose, we define two interfaces, TokenIF and SourceAccess:

```java
interface TokenIF {
  Token getNext();
}
interface SourceAccess {
  char getChar();
}
```

The new scanner component does not include the source file handling but fetches the source file characters via a required interface. We define the component type Scanner2Type with a provided interface TokenIF and a required interface SourceAccess:

```java
component type Scanner2Type {
  port token provides TokenIF;
  port source requires SourceAccess;
}
```

The Scanner2 component attaches the token port to a component fragment, a method block. It implements the TokenIF and scans the source file in order to determine the next token. When it needs the next character from the source file, it simply invokes the getChar-operation defined in the SourceAccess interface via the inside of the required port source.

```java
component Scanner2 ofType Scanner2Type {
  //port token provides TokenIF;
  //port source requires SourceAccess;
  attach This.token to TokenIF {
    Token getNext() { Token current = new Token();
      char c = This.source.getChar();
      while ( c != separator ) {
        current.append( c );
        c = This.source.getChar();
      }
      return current;
    }
  }
}
```

Figure 3: The Scanner2 component with port token providing the TokenIF implemented by a method block, and port source requiring the SourceAccess interface.

6 COMPONENT COMPOSITION

A compiler is a top-level component that is composed from a scanner, a parser etc. For that...
reason, we declare its type without any ports. The type of the parser defines a required interface TokenIF, and other ones which we do not consider.

```
component type CompilerType {}
component type ParserType {
    port ...;
    port getToken requires TokenIF;
}
```

**Subcomponents**

A component may be composed from subcomponents. E.g. the Compiler1 component (see Figure 4) is composed from a scanner, a parser, and other subcomponents like a code-generator which we disregard.

```
interface SourceFile {
    void setSource(String sourceName);
}
interface SourceHandling extends SourceFile, SourceAccess {
}
component Compiler1 ofType CompilerType {
    ParserType myParser = new Parser();
    Scanner2Type myScanner = new Scanner2();
    connect myParser.getToken to myScanner.token;
    connect myScanner.source to sourceHandler;
    attach sourceHandler to new SourceHandling{
        private File sourceFile;
        void setSource(String name) {//open sourceFile}
        char getChar() { //read next char from sourceFile}
    }
    public void main(String[] args) {
        new Compiler1();
        This.sourceHandler.setSource(sourceName);
        //start parser via a plug and port not shown
    }
}
```

**A connect-statement connects a required port of a subcomponent (instance), like getToken of Parser, to a provided port of a subcomponent (instance), like token of Scanner2, as Figure 4 shows. A constraint checked by the compiler is that a required port can be connected to only one provided port; but many required ports may be connected to the same provided port. An event port may be connected to many provided ports. The compilation of a connect-statement includes port-matching, i.e. checking if the provided port interface extends (incl. equals) the required port interface. We may use a connect-statement also to connect a port of a subcomponent directly with the inside of a matching port of the (parent) component.**

**Connecting Subcomponent Ports with Plugs**

The Compiler1 component contains a component fragment, an anonymous class implementing the interface SourceHandling, which the source port of the Scanner2 should invoke. However, a connect-statement does not allow to connect a subcomponent port with a component fragment. Therefore, we introduce plugs which replace private ports of ArchJava.

A plug is a generic construct that exceeds the generic possibilities provided by parametric interfaces or classes. The generic expression "plug<interface>" generates a plug of the interface type. It might be considered as a variable on which only a very limited set of operations may be executed: it may be used in connect- and attach-statements, or it may be used in a component fragment to invoke an operation defined in the plug interface.

The Compiler component (see Figure 4) declares a plug of the interface type SourceHandling named sourceHandler. The plug is used to pass operation invocations from the required port of the scanner subcomponent to a component fragment of the compiler component, which does all handling of the source file.

A connect-statement connects the required port source of the scanner with that plug, matching at compile time whether the plug interface extends the required port interface. The main method, which gets the filename of the source file passed as a parameter, invokes the setSource-operation via the same plug.

An attach-statement may attach a plug to a component fragment, as shown in Figure 4. It checks at compile time whether the interface of the component fragment extends the plug interface.
constraint is that the same plug may appear only once on the left-hand side of an attach- or connect-statement, but several times on their right-hand side and/or be used for operation invocations.

Factoring Out SourceHandling

Suppose that we want to reuse the anonymous source handling class with the interface SourceHandling shown in Figure 4. Then we should factor it out and transform it into a separate source file processing component with the component type SourceType.

```
component type SourceType {
    port source provides Sourcefile;
    port accessSource provides SourceAccess;
}
```

The component Source contains a SourceHandling component fragment that is identical to the component fragment used by the Compiler1 component (see Figure 4). Since we want to attach both provided ports to the same component fragment, we declare the plug sourceHandler of type SourceHandling. It is attached to the component fragment with an attach-statement. The inside of each provided port is attached to the plug with each an attach-statement.

```
component Source ofType SourceType {
    port source provides Sourcefile;
    port accessSource provides SourceAccess;
}
```

Figure 5: Component Source with the provided ports source and accessSource attached to plug sourceHandler attached to an anonymous class as component fragment.

The component Compiler2 (see Figure 6) is identical to Compiler1, except for replacing the SourceHandling component fragment by the Source component. It connects the port source of Scanner2 with a connect-statement to the accessSource port of Source. The plug setSource is declared and connected to the source port of the Source component with the objective that the main method may invoke via that plug the setSource-operation of the source port.

```
component Compiler2 ofType CompilerType {
    Parser myParser = new Parser();
    Scanner2Type myScanner = new Scanner2();
    SourceType mySource = new Source();
    connect myParser.getToken to myScanner.token;
    connect myScanner.source to mySource.accessSource;
    plug<Sourcefile> setSource;
    connect This.setSource to mySource.source;
    public void main( String[] args ) {
        String sourceName = args[1];
        new Compiler2();
        This.setSource.setSource( sourceName);
        //start parser via a plug and port not shown
    }
}
```

Figure 6: Component Compiler2 composed from subcomponents Parser, Scanner2 and Source.

7 DYNAMIC ARCHITECTURES

The language constructs described so far allow to construct component systems with a static architecture, i.e. a static hierarchy of collaborating component instances. Though that is sufficient for a large class of systems, there are other ones that require a dynamic creation and connection of components.

A component instance may be created dynamically in a method of a component fragment with a new-operator and component constructor in the same way as shown e.g. in Figure 4. Dynamically created components are connected at run-time with a reconnect-statement which is similar to a connect statement. A component should document explicitly all kinds of architectural interactions that are permitted between its subcomponents. To this purpose, a component uses connection patterns (as introduced by ArchJava (Aldrich, May 2002) (Aldrich, 2002)) to describe the set of connections that can be made at run-time using reconnect-statements.

Since in a dynamic architecture, a component may have a variable number of subcomponents of the same type, we introduce component arrays and vectors (as a parametric Vector parameterized with a component type). Since it may also be required that a connection is made from the port of a component to a variable number of sibling components, we
introduce port arrays or port vectors as arrays or parameterized vectors of an interface type.

Though the primary emphasis of component and port arrays resp. vectors is on dynamic architectures, they may be of use also for static architectures with repetitive elements.

For example, consider a WebServer component. It has one Router and many Worker subcomponents. The Router receives incoming HTTP-requests and passes them through a required port of the port array workers to the connected Worker subcomponent that serves the request. The WebServer starts the Router via its provided port start and the plug start.

Figure 7 shows a shortened version of the WebServer. The running version with about three times the length of the presented version may be obtained from the authors. We present, in contrast to (Aldrich, May 2002), an optimized solution that reuses idle Worker instances and their connections. A Worker contains a WorkerThread class. When an httpRequest is invoked via the serve port of a Worker, the WorkerThread is (re-) started by a notify-statement and takes up work with a call of its method handleRequest. When it has finished the processing of an HTTP-request, it goes into a wait state.

The WebServer has declared an array of Worker components. It connects the provided serve port of each Worker instance after its creation dynamically to the matching port of the required port array workers of the Router component.

The WebServer performs the administration of the Worker instances in the method block implementing the WorkerAdministration interface, which is attached to the adminWorker plug. It has a setIdle-operation which is invoked by a Worker after having finished the processing of an HTTP-request, and similar operations. The requestWorker-operation checks if an idle Worker is available, and returns its index. Otherwise, it creates a new Worker instance if the maximum worker number is not yet reached. It connects dynamically a Worker’s serve port to the matching port of the workers port array of the Router, and its required adminWorker port to the adminWorker plug.

The WebServer has connected the required request port of the Router to the adminWorker plug. In that way, both the Router and all Worker’s can invoke operations of the worker administration, like setIdle or requestWorker when required.

The code of the WebServer component is easy to understand, in contrast to the code shown in (Aldrich, May 2002).
CompJava has been extended for use as a distributed component language as described in (Schmid, 2005).

REFERENCES

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995 Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.

8 CONCLUSIONS

CompJava, to be available for a wider use in fall 2006 via http://www-home.fh-konstanz.de/~schmidha/, composes components in a clear and simple way from two kinds of building blocks: component fragments and subcomponents. We have introduced component fragments that may be considered as very simply structured lightweight components without ports. There are three implementation variants covering different performance and reusability requirements. Component fragments allow to structure low-level components in an adequate way, and they serve as filters for medium to high level components.

These building blocks with well-defined and clear interfaces are attached/connected either directly or via plugs to themselves or to ports of the parent component.

Clean and efficient dynamic architectures are composed from dynamically instantiated and connected subcomponent instances together with component arrays and port arrays resp. vectors.
EXPLORING FEASIBILITY OF SOFTWARE DEFECTS
ORTHOGONAL CLASSIFICATION

Davide Falessi, Giovanni Cantone
Univ. of Roma "Tor Vergata", DISP, Via del Politecnico 1, Rome, Italy
falessi@ing.uniroma2.it, cantone@uniroma2.it

Keywords: Software engineering, Experimental software engineering, Orthogonal Defect Classification, Defect class affinity, Fault detection, Effectiveness, Efficiency.

Abstract: Defect categorization is the basis of many works that relate to software defect detection. The assumption is that different subjects assign the same category to the same defect. Because this assumption was questioned, our following decision was to study the phenomenon, in the aim of providing empirical evidence. Because defects can be categorized by using different criteria, and the experience of the involved professionals in using such a criterion could affect the results, our further decisions were: (i) to focus on the IBM Orthogonal Defect Classification (ODC); (ii) to involve professionals after having stabilized process and materials with students. This paper is concerned with our basic experiment. We analyze a benchmark including two thousand and more data that we achieved through twenty-four segments of code, each segment seeded with one defect, and by one hundred twelve sophomores, trained for six hours, and then assigned to classify those defects in a controlled environment for three continual hours. The focus is on: Discrepancy among categorizers, and orthogonality, affinity, effectiveness, and efficiency of categorizations. Results show: (i) training is necessary to achieve orthogonal and effective classifications, and obtain agreement between subjects, (ii) efficiency is five minutes per defect classification in the average, (iii) there is affinity between some categories.

1 INTRODUCTION

Defect classification plays an important role in software quality. In fact, software quality is strictly related to the number and types of defects present in software artifacts and eventually in software code.

The analysis of defect data can help to better understand the quality of software products and the related processes, and how they evolve.

An invalid defect categorization would obviously imply wrong data, which could lead analysts to wrong conclusions, concerning the product, development process or phase, methods, and/or tools.

For instance, in order to define the best mix of code testing and inspection techniques for given application domain and development environment, it is crucial to collect valid defect-category data (Basili & Selby, 1987; Cantone et al., 2003; Abdelnabi et al., 2004).

1.1 Related Works

The Orthogonal Defect Classification (ODC) is a schema (IBMa, 2006) that IBM proposed in the aim of capturing semantics of software defects (see Section 1.3 for further details concerning ODC). ODC was originally published on 1992; because in the mean time the software world changed, the IBM provided to update ODC regularly. The classification adopted in this work is ODC v5.11, i.e. the last version of ODC, to the best of our knowledge. ODC is defined as a technology-independent (software process, programming language, operative system, etc.) classification schema. This is based on eight different kinds of attributes, each of them having its own categories.

Khaled El Emam and Isabella Wieczorek (1960), and Kennet Henningsson and Claes Wohlin (2004) investigated ODC empirically by focusing on subjectivity of defect classification. In order to evaluate the level of cohesion among classifications that different subjects enacted, both studies used “Kappa statistics” (Cohen, 1960), and worked on
their own variations of ODC. In particular, El Emam and Wieczorek involved various combinations of three subjects who performed in the role of defect categorizers on an actual software artifact, during the development process; they hence collected and eventually analyzed “real inspection data” (El Emam & Wieczorek, 1960). Eight subjects, each having at least a Master’s degree but with limited experience in defect classification, participated to the experiments conducted by Henningsson and Wohlin, where objects were utilized that included thirty defects selected from a repository. Concerning results from those studies, the former presents high level of cohesion with respect to standards utilized by medical studies, the latter shows that there might be subjectivity in classification. Durães and Madeira (2003) used the ODC as initial defect categorization framework and afterwards faults were classified in a detailed manner according to the high-level constructs where the faults reside and their effects in the program. The analysis of field data on more than five hundred real software faults shows a clear trend in fault distribution across ODC classes. Moreover, results show that a smaller subset of specific fault types is clearly dominant regarding fault occurrence.

1.2 Study Motivations and View

We can count a significant number of empirical works from many authors worldwide, whose conclusions are based on categorization of software defects. A common assumption of all those works (see Section 8 for few samples of them: (Basili & Selby, 1987; Cantone et al., 2003; Juristo and Vegas, 2003; Myers, 1978)) is that in large extent defects can be classified objectively, whatever the classification model might be. In the absence of enough evidence for such an assumption, all those empirical results could be questioned. Consequently, the basic question of this study is whether software practitioners can uniformly categorize defects.

In this paper we focus on the ODC attribute “Defect Type” (DT), which role is to catch the semantics of defects, that is the nature of the actual correction that was made to remove a defect from a software code. DT categorization hence follows defect detection, identification and fixing: in fact, the real nature of a defect can be understood (and than suitably categorized) only after the code is fixed, in the ODC approach.

DT includes seven defect categories (IBMa, 2006; IBMb, 2006):

1. Assignment/Initialization: value(s) assigned incorrectly or not assigned at all.
2. Checking: errors caused by missing or incorrect validation of parameters or data in conditional statements. It might be expected that a consequence of checking for a value would require additional code such as a do while loop or branch.
3. Algorithm/Method: efficiency or correctness problems that affect the task and can be fixed by re-implementing an algorithm or local data structure without the need for requesting a design change; problems in the procedure, template, or overloaded function that describes a service offered by an object.
4. Function/Class/Object: the defect should require a formal design change, as it affects significantly capability, end-user interfaces, product interfaces, interface with hardware architecture, or global data structure(s); defect occurred when implementing the state and capabilities of a real or an abstract entity.
5. Interface/O-O Messages: communication problems between modules, components, device drivers, objects or functions.
6. Relationship: problems related to associations among procedures, data structures and objects.
7. Timing/Serialization: necessary serialization of shared resource was missing, the wrong resource was serialized, or the wrong serialization technique was employed.

In the remaining, we present, analyze, and discuss a benchmark including two-thousand and more data that we achieved through an experiment based on twenty-four segments of code, each segment seeded with one defect, and one hundred twelve sophomores, trained for six hours and then assigned to classify those defects in a controlled environment for three continual hours. In particular, Section 2 presents the experiment problem and goal definition. Section 3 shows the experiment planning and operation. Section 4 and 5 present and discuss results. Some final remarks and further intended works conclude the paper.

2 GOAL AND EXPERIMENT HYPOTHESES

The goal (Basili et al., 1987) of this paper is to analyze the (ODC)’s DT attribute from the point of view of the researcher, in the context of an academic course on “OO thinking and programming with Java” for sophomores, for the purpose of evaluating dependences of software defect categorizations on:
i) defect \((d \in DD)\): ii) subjectivity of practitioners \((s \in S)\); iii) expertise in defect detection \((X)\), and (iv) Programming language \((PL)\) utilized to code artifacts, by focusing on: a) Effectiveness \((E)\), i.e. in what extent a defect is associated to its most frequent categorization \((MFC)\); b) Efficiency \((Ec)\), i.e. the number of \((MFC)s\) per time unit; c) Orthogonality \((O)\), i.e. in what extent a defect is assigned to just one category; d) Affinity \((A)\), i.e. in what extent a defect category looks like other categories, and e) Discrepancy \((D)\), i.e. in what extent subjects assign a defect different categories (see Sections 3 for quantitative definitions of all those variables).

Based on that goal, the hypotheses of our work concern the impact of expertise \((h_x)\), defect category \((h_c)\), and programming language \((h_{pl})\) on orthogonality \((h_{o})\), effectiveness \((h_{e})\), and discrepancy \((h_{d})\).

The null \((h_0)\) and alternative \((h_1)\) hypotheses for expertise versus orthogonality \((resp.\ effectiveness,\ and\ discrepancy)\) are:

- \(h_{X0}\): Expertise does not significantly impact on orthogonality \((resp.\ h_{E0},\ and\ h_{D0})\).
- \(h_{X1}\): Expertise impacts significantly on orthogonality \((resp.\ h_{E1},\ and\ h_{D1})\).

Hypotheses concerning programming language \((h_{L0}, h_{L1}, h_{E0}, h_{E1}, h_{D0}, h_{D1})\), and defect category \((h_{C0}, h_{C1}, h_{E0}, h_{E1}, h_{D0}, h_{D1})\) have similar formulations. In the remaining, while we evaluate the impact of defect category, expertise, and programming language on outcomes, our reasoning mainly focuses on expertise. In fact, in our expectation, in case of significant dependence of defect categorizations from the categorizers’ subjectivity, expertise should play the most important role and behave as the main discriminating factor; consequently, our planning and training emphasis was in providing variable expertise.

3 EXPERIMENT PLANNING AND OPERATION

Whoever the participant subject, three items characterize our elementary experiment: a defect, as seeded and fixed in a program segment, the programming language of that segment, and dissimilarity of that defect.

In order to average on differences among participant subjects, our planning decisions was to utilize subjects with the same level of experience; in particular: i) one hundred or more subjects from the same academic class, ii) subjects showing the same OOP class frequency record, iii) subjects who would be attending all the training sessions. Moreover, in order to manage the impact of learning effect on results, we kept further planning decisions, which also helped to prevent exchange of information among participant subjects: iv) to arrange four master files, where experiment artifacts are located in different order, v) to assign subjects seats randomly, and give neighbors copies of different master files, and vii) to ask subjects to handle artifacts in sequence, staring from the first artifact their assigned.

We hence developed and saved into repository defected artifacts. An artifact consists in a less than twenty ELOC segment of code, plus comments to ensure easy and valid understanding; one defect is seeded per code segment, and fixed through specific comments. Let us note that while we used our understanding of DT ODC to generate defected artifacts, we no further utilize such understanding in the remaining of this study, where categorizations are utilized as enacted by subjects.

In parallel with repository construction, we called for participation, and trained subjects through three two-hour lectures, which presented the role and importance of defect categorization, defined categories of the ODC DT attribute, and explained extensively two or more exemplar cases for each defect category. Subsequently, we evaluated in Low \((L)\), Average \((A)\), and High \((H)\) the dissimilarity between defects in the experiment artifacts and defects in the examples given for training (see Expertise in Section 3.1.4 for further details).

Finally, we ruled the random selection of experiment artifacts from the repository, as in the following: (i) Get as many C++ as Java coded artifacts; (ii) Get two or more artifacts for each defect category; (iii) Get 20% of artifacts for each value of Dissimilarity, and remaining (40%) at random.

3.1 Independent Variables: Parameters, Blocking Variables and Factors

3.1.1 Subjects

As already mentioned, one hundred twelve sophomores participated to the experiment, who were attending the course of Object-Oriented Programming, their fourth CS course at least. All of them had attended all the training lectures and, in term of experience, they can be considered as novice.
programmers. Subjects’ participation was part of a course test; they worked individually in the same 250 seats room, in the continual presence of two or more observers; communication among subjects was not allowed. Other one hundred subjects, who had not fully attended the training or the OOP course, were located in an adjacent room: their data will be no further considered in the present paper.

3.1.2 Objects

Experiment artifacts, twelve C++ coded and twelve Java coded, were assigned to all subjects, each artifact seeded by one defect. All quadruples of neighbor subjects handled the same artifacts but in different order.

3.1.3 Experiment Duration

Subjects had up to three hours assigned to enact their task. They were allowed to quit the experiment any time, after the start and before the formal end.

3.1.4 Factors and Treatments

Factors of the basic experiment and their levels are:

- **Programming Language (PL)**, levelled at C++ and Java, respectively.

- **Defect Category (Ctg)**. Six defect categories are utilized, i.e. all the DT ODC less Timing/Serialization: in fact, subjects had not yet been exposed to concurrent programming concepts, constructs, and mechanisms, when they participated to the experiment.

- **Expertise (X)**. It is analogous to Dissimilarity but scale is reversed; it hence relates to quantity of examples given per defect during training. In fact, for each defect type, we set artificially the subjects expertise by dosing the explanation time, and the numbers of examples given per defect. (0, 1, 2) are the values of the ordinal scale we use to measure the subjects expertise, where: 0 means that training did not include examples showing that specific instance of the defect category (hence, the defect shows low level of similarity with the explained defects, and its Dissimilarity measure is H); 1 means that training exposed subjects just one time to that specific instance of the defect category (Dissimilarity measures A); 2, means that subjects trained with two or more instances concerning that specific defect category (the defect shows high similarity with the explained ones, and its Dissimilarity measure is L). Concerning this point, let us finally note that, because subjects had already attended two CS courses in C++ and were attending a Java course, trainers gave more emphasis to defected artifacts coded in the latter.

3.2 Dependent Variables

We directly measured:

- **Completion Time**: Actual task duration per subject (duration of all the elementary experiments assigned to the same subject).

- **Categorization**: ODC per elementary experiment and subject. A subject, whether sure about his understanding, assigns a defect just one category, else zero or two categories.

Based on such direct measures, we derive the variables described in the followings, which characterize the DT attributes of the OD Classification. Let us note that measures in the following are given to each specific defect, and then applied in the same way in each defect category, each programming language, and so on.

- **Effectiveness (E)**: percentage of the most frequent categorization with respect to the universe of categorizations given by subjects for this defect.

- **Efficiency (Ec)**: how many (MFC)s occur per time unit, in the average, for this defect. Because of the experiment infrastructure that we choose (paper supports for data collection; data registration enacted by subjects), our decision was to collect the task Completion time only, rather than the time duration of each elementary experiment. Consequently, data from the basic experiment are not enough to investigate efficiency in deep.

- **Orthogonality (O)**: what percentage of subjects assigned this defect just one category (rather than zero or two).

- **Discrepancy (D)**: this does measure the average distance in percentage related to the entire population for the same categorization, and is a variant of the Agreement’s (Henningsson and Wohlin, 2004; El Emam & Wieczorek, 1960) one complement. In other word, discrepancy is the average probability that a given categorization is different from those given by other subjects for the same defect.

- **Affinity (A)**: this expresses a relationship of a category with respect to one more category, and is a variant of the Confusion’s (Henningsson and Wohlin, 2004) one complement. Given two categories, the source category CS and the
destination category CD, let us take into account defects, which MFC is CS. The affinity of CS with respect to CD, \( A_{\text{WRT}}(CS, CD) \), measures the percentage of CS or CD categorizations given for those defects. Formally:

\[
\forall d \in \text{DD(Exp)}: \text{MFC}(d) = \text{CS}, \exists A_{\text{WRT}} \in [0..100]: (100*\text{p}(d) \in \{\text{CS, CD}\} = A_{\text{WRT}}); \tag{1}
\]

where: \( d \) is any of the defect set DD in the experiment Exp, and \( \text{p} \) is the probability function averaged on all instances of the argument defect. \( A_{\text{WRT}} \) is not commutative (sometimes \( A_{\text{WRT}}(C_1, C_2) \neq A_{\text{WRT}}(C_1, C_2) \)), and its reflexive closure, \( A_{\text{WRT}}(C, C) \), is the Effectiveness with respect to category C.

The affinity between CS and CD, \( A_{\text{Btw}}(CS, CD) \), is then defined as:

\[
\forall d \in \text{DD(Exp)}: \text{MFC}(d) \in \{\text{CS, CD}\}, \exists A_{\text{Btw}} \in [0..100]: (100*\text{p}(d) \in \{\text{CS, CD}\} = A_{\text{Btw}}); \tag{2}
\]

Note that \( \forall (CS, CD), A_{\text{Btw}}(CS, CD) = A_{\text{Btw}}(CD, CS) \), i.e. \( A_{\text{Btw}} \) is commutative.

Definitions above can be extended to three or more categories.

4 RESULTS AND DATA ANALYSIS

At experiment conduction time, subjects registered more than two thousand six hundred data fields, which we eventually deposited in a database. Two subjects provided exorbitantly distant data from the most frequent ones; data analysis identified those data as outliers, and consequently we excluded them from further analysis.

In this study all categorizations given by subjects, are evaluated, null ones included: in our evaluation, null categorizations candidate IBM DT definitions for further clarification, or our training for improvement.

4.1 Descriptive Statistics

Let us consider now orderly relationships between each response variable and factors.

4.1.1 Effectiveness

We want to describe the evolution of the most frequent categorizations as a whole and versus expertise, programming languages, and defect categories involved, and eventually with respect to the task completion time.

Figure 1 shows subjects given categorizations, as averaged on the whole available data. Concerning the abscissa, “0” stands for not categorized defects (null); “1_MFC” (resp. “1_NMFC”) denotes that the subject assigned this defect just the most frequent categorization (resp, one category, but different from the MFC); “Others” stands for assignment of two categories to this defect. Effectiveness (see MFC in Figure 1) is 0.69, and variance is 8.

Figure 2 and Figure 3 relate effectiveness with expertise and specific defects, respectively.

Table 1 shows effectiveness versus ODC categories, and related variances. Table 2 relates effectiveness to the programming language of the defected segments.

Figure 4 shows the evolution of effectiveness in time.

4.1.2 Efficiency

Figure 5 presents efficiency with respect to completion time. Table 3 shows statistical summary for efficiency.

4.1.3 Orthogonality

Figure 6 and Figure 7 relate orthogonality with expertise and specific defects, respectively. Table 4 and Table 5 present orthogonality versus ODC categories, and programming language, respectively.

Table 6 shows statistical summary for Orthogonality, and Figure 8 presents the evolution of orthogonality in time.

![Figure 1: Categorizations and Effectiveness (with respect to the whole data collected).](image-url)
Figure 2: Effectiveness versus Expertise.

Figure 3: Effectiveness per Defect.

Table 1: Effectiveness versus ODC Categories.

<table>
<thead>
<tr>
<th>Category</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average (%)</td>
<td>77</td>
<td>83</td>
<td>48</td>
<td>75</td>
<td>54</td>
<td>82</td>
</tr>
<tr>
<td>Variance</td>
<td>204</td>
<td>156</td>
<td>18</td>
<td>470</td>
<td>237</td>
<td>151</td>
</tr>
</tbody>
</table>

Table 2: Effectiveness versus Programming Language.

<table>
<thead>
<tr>
<th>Language</th>
<th>Java</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average (%)</td>
<td>61</td>
<td>78</td>
</tr>
<tr>
<td>Variance</td>
<td>402</td>
<td>248</td>
</tr>
</tbody>
</table>

Figure 4: Effectiveness in time.

Figure 5: Efficiency in time.

Table 3: Statistical summary for efficiency.

<table>
<thead>
<tr>
<th>Efficiency</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average (MFC/h)</td>
<td>9</td>
</tr>
<tr>
<td>Variance</td>
<td>7,31</td>
</tr>
</tbody>
</table>

Figure 6: Orthogonality versus Expertise.
Table 4: Orthogonality versus ODC categories.

<table>
<thead>
<tr>
<th>Category</th>
<th>Orthogonality</th>
<th>Average (%)</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>98</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>97</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>94</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>98</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>95</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>98</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5: Orthogonality versus programming language.

<table>
<thead>
<tr>
<th>Language</th>
<th>Orthogonality</th>
<th>Average (%)</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td></td>
<td>96</td>
<td>6</td>
</tr>
<tr>
<td>C++</td>
<td></td>
<td>97</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 6: Statistical summary for orthogonality.

<table>
<thead>
<tr>
<th>Orthogonality</th>
<th>Average (%)</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>97</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 7: Statistical summary for Discrepancy.

<table>
<thead>
<tr>
<th>Discrepancy</th>
<th>Average (%)</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>43</td>
<td>1010</td>
</tr>
</tbody>
</table>

Table 8: Discrepancy versus ODC Categories.

<table>
<thead>
<tr>
<th>Category</th>
<th>Discrepancy</th>
<th>Average (%)</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>39</td>
<td>44</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>28</td>
<td>379</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>65</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>34</td>
<td>591</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>60</td>
<td>171</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>29</td>
<td>326</td>
</tr>
</tbody>
</table>

Table 9: Discrepancy versus Programming Language.

<table>
<thead>
<tr>
<th>Language</th>
<th>Discrepancy</th>
<th>Average (%)</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td></td>
<td>50</td>
<td>453</td>
</tr>
<tr>
<td>C++</td>
<td></td>
<td>50</td>
<td>478</td>
</tr>
</tbody>
</table>

Figure 7: Orthogonality per defect.

Figure 8: Orthogonality in time.

4.1.4 Discrepancy

Table 7 shows statistical summary for discrepancy. In the remaining, this Section presents discrepancy with respect to ODC categories (Table 8), programming languages (Table 9), expertise (Figure 9), and seeded defects (Figure 10), respectively.

Figure 9: Discrepancy versus Expertise.

Figure 10: Discrepancy per defect.

4.1.5 Affinity

Based on the average effectiveness shown above (E=0.69), the number of categorizations that differ from their (MFC)s is around 818.
While it is not possible to include all those categorizations in this paper, we can describe their tendencies, based on definitions given for Affinity in Section 3.2 above: according to expressions (1), \( A_{WRT}(6, 5) = 90 \); \( A_{WRT}(2, 3) = 95 \); according to expression (2), \( A_{Btw}(1, 3, 5) = 90 \).

In words, when the MFC is 6 (Relationship) then 90% of categorizations provided by subjects are 6 (Relationship) or 5 (Interface/ OO Messages).

Moreover, when the MFC is 2 (Checking) then 95% of categorizations provided by subjects are 2 (Checking) or 3 (Algorithm/Method).

Furthermore, when MFC is 1 (Assignment/Initialization), 3, or 5 then 90% of categorizations provided by subjects are 1, 3, or 5.

Finally, let us spread on data “Others” in Figure 1, which concern affinity. Columns Ctg1 and Ctg2 in Table 10 present the alternative categorizations that doubtful subjects assigned to defects; the Ocs columns show the occurrences of those double categorizations.

Table 10: Defect’s double categorizations (as provided by doubtful subjects).

<table>
<thead>
<tr>
<th>Ctg1</th>
<th>Ctg2</th>
<th>Ocs</th>
<th>Ctg1</th>
<th>Ctg2</th>
<th>Ocs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>Others</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

4.2 Hypothesis Testing

In order to test hypotheses concerning expertise, we separate cases where the involved expertise is null (0) from remaining ones (expertise measures 1 or 2), so having the seeded defects partitioned in two groups, \( G_{X=0} \) and \( G_{X\neq0} \), respectively.

4.2.1 Testing \( h_{X=0} \): Expertise does Insignificantly Impact on Orthogonality: \( O(G_{X=0}) \not\equiv O(G_{X\neq0}) \)

The number of subjects, who assigned one category to \( G_{X=0} \) defects, are: (100, 101, 103, 104, 104, 105, 105, 106, 106, 107, 108), respectively; those for \( G_{X\neq0} \) are: (103, 106, 107, 108, 109, 109, 109, 109, 110, 110, 110, 110). Figure 11 shows the Box-and-Whisker plots for such series of data. Since the latter cannot fit under normal curve at 95% of confidence level (in fact, its lowest P-value from Shapiro-Wilks test is 0.037, which is less than 0.05), we applied the W test. Since the W test’s P-value is 0.000194, which is less than 0.05, there is a statistically significant difference between the medians at the 95.0% confidence level. Consequently, we can reject the null hypothesis \( h_{X=0} \) at 95% of significance level. In other words, expertise significantly impacts on orthogonality of defect categorizations.

4.2.2 Testing \( h_{X\neq0} \): Expertise does Insignificantly Impact on Effectiveness: \( E(G_{X=0}) \not\equiv E(G_{X\neq0}) \)

The effectiveness values for categorizing \( G_{X=0} \) defects are (41, 41, 45, 46, 52, 54, 59, 65, 71, 75, 79, 81), respectively; those for \( G_{X\neq0} \) are (74, 77, 79, 81, 83, 98, 100, 104, 105, 107, 108, 109). Figure 12 shows the Box-and-Whisker plots for such series of data. Since the latter cannot fit under normal curve at 95% of confidence level (in fact, its lowest P-value from Shapiro-Wilks test is 0.037, which is less than 0.05), we applied the W test. Since the W test’s P-value is 0.000194, which is less than 0.05, there is a statistically significant difference between the medians at the 95.0% confidence level. Consequently, we can reject the null hypothesis \( h_{X\neq0} \) at 95% of significance level. In other words, expertise significantly impacts on effectiveness of defect categorizations.
4.2.3 Testing $h_{XD0}$. Expertise does Insignificantly Impact on Discrepancy: $D(G_{X=0}) \cong D(G_{X\neq0})$

The discrepancy values related to categorizations of $G_{X=0}$ defects are (42, 44, 49, 52, 58, 59, 63, 68, 68, 70, 71, 74), respectively; those for $G_{X\neq0}$ are (2, 4, 5, 9, 10, 17, 19, 40, 41, 42, 46, 51). Figure 13 shows the Box-and-Whisker plots for such series of data. Since the latter cannot fit under normal curve (in fact, its lowest P-value from Chi-Square test is 0.022, which is less than 0.05), we applied the W test. Since the W test’s P-value is 0.000137, which is less than 0.05, there is a statistically significant difference between the medians at the 95.0% confidence level. Consequently, we can reject the null hypothesis $h_{XD0}$ at 95% of significance level. In other words, expertise significantly impacts on discrepancy between defect categorizations.

![Box-and-Whisker plots](image)

Figure 13: Level of Discrepancy, treated by experience.

5 DISCUSSION

5.1 Experiment Results

5.1.1 Effectiveness

Based on Figure 1, the percentage of most frequent categorizations is in average 69%. This seems quite a small value for effectiveness, which also means that there seems to be high subjectivity in defect categorization when trained/untrained novice programmers are involved. Again Figure 1 shows that those programmers perform quite dissimilarly, since variance (8) is very high - one third of the seeded defects (24) - as also shown by Figure 3 and Table 1.

Figure 1 also shows that single non-MFC classifications (1_NMFC) are in number ten times greater than the doubtful ones (Null + Others). In our understanding, this means that novices seem unconscious of consequences that their limited knowledge of ODC DT could have. Another view is that IBM should improve the presentation of ODC DT, in order to help practitioners to distinguish among categories more easily.

Based on Figure 2, it seems that effectiveness is strongly related to expertise. In fact, effectiveness grows from 54% up to 89% as the given training grows. Based on that slope, the trend for effectiveness is 100%, which expert professionals should be able to approach. The impact of expertise on results explains, in our understanding, the variance previously observed with aggregated data. This also asserts that data in Table 1 should not be utilized to evaluate the impact of defect category on effectiveness, and, similarly, data in Table 2 should not be used to evaluate the impact of programming languages on effectiveness.

Finally, based on Figure 4, effectiveness seems independent from the completion time, when this is limited to 3 hours.

5.1.2 Efficiency

The amount of time a subject employed to enact a categorization is around 5 minutes in average.

Based on date in Table 3, the mean time for an MFC categorization is 6.66 minutes (9 MFC/hour), and variance is 7.3 MFC/hour.

Since variance is similar to the average, it seems that efficiency is highly subjective with novice programmers. Let us recall that it was not possible to collect the duration time during the basic experiment; consequently, we cannot investigate efficiency more deeply.

5.1.3 Orthogonality

Based on Table 6, which data are again not yet disaggregated with respect to expertise, orthogonality is 97%, while variance is 2. This expresses that, in the average, programmers commonly percept ODC with respect, and tend to provide just one classification per defect, whatever is their expertise. However, taking in consideration data disaggregated by expertise (Figure 6), with novices, orthogonality grows from 95% up to 99.3% as expertise grows.

Based on Table 5, aggregated data show no difference of C++ and Java versus orthogonality.

Finally, based on Figure 8, orthogonality seems independent from the completion time, when this is limited to 3 hours.
5.1.4 Discrepancy

Subjects had to select a category out of seven (including null). In theory, the maximum value for discrepancy is 86%, which occurs when all selections are equally probable; it is the probability that six categories are selected out seven (less scale factor 100). The minimum of discrepancy is 0%, which occurs in case of complete agreement between subjects for each categorization. Table 7 shows 43% discrepancy (and 1010 variance!), as registered in average for our basic experiment, again with respect to data not yet disaggregated by expertise. That value is exactly the mean between the discrepancy’s minimum and maximum theoretic values; as a result, ODC seems to be quite dependent from the categorizers’ subjectivity, when trained/untrained novices are involved.

Based on Figure 9, which relates to data disaggregated by expertise, it seems that discrepancy is strongly related to expertise. In fact, discrepancy decreases from 60% up to 17% as the given training grows. Based on that slope, the trend for discrepancy is the theoretic minimum (0%), which expert professionals should be able to approach. The impact of expertise on results explains, in our understanding, the very large value previously observed for variance, when aggregated data were considered.

Based on Table 9, aggregated data show no difference of C++ and Java versus discrepancy. Again, discrepancy seems independent from the completion time, when this is limited to 3 hours.

5.1.5 Affinity

Based on data elaboration that we presented above (see Section 4.1.5), it seems that categories “Assignment/ Initialization”, “Algorithm/Method”, and “Interface/OO Message” are one each other strongly affine. Moreover, category “Interface/OO Message” is frequently provided in place of “Relationship”, and the same for “Algorithm/Method” with respect to “Checking”. This, in our understanding, calls for training improvement by emphasizing on dissimilarities among those categories.

5.2 Threats to Validity

This empirical study has a number of limitations that should be taken into account when interpreting its results.

Concerning the internal validity (Wohlin et al., 1978) (i.e. the degree to which conclusions can be drawn about the causal relationship between independent variables and dependent variables), it should be noted that we utilized a very limited number of defect samples: 12 per language, hence two defects per category. Moreover, while the task completion time assigned was quite small, and subjects were continually in control of observers during the conduction of the experiment, we cannot guarantee absence of interactions between participants; in fact, these were student, who we partially graded for their performance; in the experiment cultural context, a student is appreciated, who passes his solutions to colleagues. Furthermore, our training emphasized on Java language, and the real experience and expertise of subjects with C++ was not in control.

Another limitation of this study is related to the external validity (Wohlin et al., 1978), i.e. the degree to which the results from this study can be generalized. It cannot be assumed a priori that the results of a study generalize beyond the specific environment and context in which it was conducted. In fact, subjects involved with the basic experiment are sophomores in OO Programming, who should not be considered as novice professional programmers. Moreover, the experiment software artifacts that we utilized in the basic experiment are small segments of code, which should not be taken to represent real software. Finally, we utilized paper supports both for experiment artifacts and forms, while realism asked for electronic-supported code, and electronic-network-supported form distribution, and data collection.

6 CONCLUSIONS AND FUTURE WORKS

This paper has presented an empirical investigation on the (IBM)’s ODC-DT attribute for software defect categorization. Foci of the investigation have been the classification effectiveness, efficiency, orthogonality, discrepancy, and affinity with respect to practitioners’ subjectivity (110 students performing in the role of experiment subjects), defects individuality (6 DT categories of seeded defects), and software artifacts’ coding language (Java and C++). Results shown include averages for time for defect categorization (5 minutes), effectiveness (69%), and orthogonality (97%). Results also show that subject’s expertise seems to
impact very significantly on all the results, and subjects with enough expertise should be able to easily approach the theoretic best value for effectiveness, as for orthogonality and discrepancy. Our consequent expectation is that there should be objectivity in defect categorization, whether enacted by software practitioners. However, such an expectation still needs empirical evidence. Further results show that, when time spent in categorizing defects lasts between 1 and 3 hours, the effectiveness, orthogonality, and discrepancy are not affected by the time duration of the classification section. Moreover, results show that the programming language of coded artifacts, and the defect nature seem to impact insignificantly on effectiveness, orthogonality, and discrepancy. Finally, our results show that there are some categories that tend to confuse subjects; this, in our understanding, calls for improving definitions of those ODC DT categories, as actually given by IBM. Namely, those categories are “Interface/OO Message” and “Relationships”. Further confusing categories are “Assignment/Initialization” and “Algorithm/Method” on one side, and “Algorithm/Method” and “Checking” on the other side, which confirm previous results (Henningsson and Wohlin, 2004).

Our plan for the future is first to extend the size of our defect repository, place the material in electronic format, and contact IBM experts in the aim of receiving their categorizations of our defect samples (to use as the reference “correct” categorizations), and then to proceed with replicating the experiment with professionals both in a controlled environment, and through the Web. This should also provide the precise timing of each categorization, and help to investigate efficiency in deep.

REFERENCES


MDE FOR BPM
A Systematic Review

Jose Manuel Perez
UCLM-Soluziona Research and Development Institute, Ronda de Toledo s/n, 13005, Ciudad Real, Spain
Josem.Perez2@alu.uclm.es

Francisco Ruiz, Mario Piattini
Alarcos Research Group, University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain
Francisco.RuizG@uclm.es, Mario.Piattini@uclm.es

Keywords: Business process management, Model driven engineering, Model driven architecture, Systematic review.

Abstract: Due to the rapid change in the business processes of organizations, Business Process Management (BPM) has come into being. BPM helps business analysts to manage all concerns related to business processes, but the gap between these analysts and people who build the applications is still large. The organization’s value chain changes very rapidly; to modify simultaneously the systems that support the business management process is impossible. MDE (Model Driven Engineering) is a good support for transferring these business process changes to the systems that implement these processes. Thus, by using any MDE approach, such as MDA, the alignment between business people and software engineering should be improved. To discover the different proposals that exist in this area, a systematic review was performed. As a result, the OMG’s Business Process Definition Metamodel (BPDM) has been identified as the standard that will be the key for the application of MDA for BPM.

1 INTRODUCTION

There is a need for today’s business to create and modify value chains rapidly. This brings about continuous growth and change in business processes. The goal of Business Process Management (BPM) is to help business people to manage these changes.

Business process management is defined as the capability to discover, design, deploy, execute, interact, operate, optimize and analyze process in a way that is complete, doing it at the business design level and not at the technical implementation level (Smith, et al., 2002).

BPM offers numerous benefits to organizations such as improving the speed of business, giving increased customer satisfaction, process integrity and accountability. It promotes process optimization, at the same time eliminating unnecessary tasks. It also includes customers and partners alike in the business processes and provides organizational agility.

BPM represents a “third wave” in business process engineering. The first wave was guided by process papers that reorganized human activity. The second wave focused on reengineering of business processes and the use of Enterprise Resource Planning (ERP). The third wave centers on formal business process models and the ability to modify and combine those models so as to align business process with organizational needs (Frankel, 2003).

BPM starts with process modeling. Process modeling is a business-driven exercise in which current and proposed process flows are documented in detail, linked to quantifiable performance metrics, and optimized through simulation analysis. Standards for process modeling languages are the key to the attaining of BPM’s goal as well as in achieving the platform independence of the process models. Platform independence is one of the principles on which Model Driven Engineering (MDE) is based. The combination of both concepts, MDE and BPM, is the target of this systematic review.

MDE was conceived in an effort to solve several problems that have arisen in the last decade. On one hand, the growth of platform complexity, there being thousands of classes and methods with very complicated dependencies. On the other hand, we
can observe the continuous technological evolution of the systems, forcing programmers to modify the system code every time a new requirement is given.

In the MDE paradigm, every concept must be modeled. Thus, any change in the system must be shown in the model that represents that system. To model the systems, MDE proposes using Domain-Specific Modeling Languages (DSML). By means of these languages, different modeling notations for each kind of system are achieved. Thus, the software engineer has specific tools for modeling all kind of systems.

Another important concept in MDE is model transformation. By transforming models, the evolution of the systems is facilitated. A model could be transformed to another model or to a XML specification as well as to the source code that implements the model functionality.

The OMG group has developed Model Driven Architecture (MDA) as an example of MDE. MDA emerged with the established idea of separating the business logic specification of a system from the platform specific details in which the system is implemented (Miller, et al. 2003). MDA adds some concepts to the MDE philosophy. MDA defines three level of abstraction. The Computational Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM).

The key technology in MDA is MOF, as it is as in the definition of metamodels, which are MOF instances (figure 1) (Bézivin, 2003). The transformations among these models are the basis of MDA philosophy.

![Figure 1: MOF metamodels structure](image)

The structure of this paper is as follows. In section 2, systematic reviews are introduced. In section 3, the carrying out of the review is shown in part, presenting the selection of studies and the classification of these. The information analysis is described in section 4 by summarizing the different authors’ proposals about the MDE for BPM application. Section 5 presents the conclusions extracted from the systematic review along with future work, taking into account the different views found.

## 2 SYSTEMATIC REVIEWS

A systematic review of the literature is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest (Kitchenham, 2004).

Systematic review is a scientific methodology that can be used to integrate empirical research on software engineering (Travassos, 2005).

Some of the characteristics that make the above methodology different from a conventional review are that a systematic review starts by defining a review protocol that specifies the research question, along with the methods and the criteria to drive the review. Added to all this, a systematic review is based on a search strategy that aims to detect as much relevant literature as possible. Moreover, performing a systematic review is needed in order to document the whole search strategy so that another researcher can replicate the same review with identical results.

There are three main phases that organize the different stages of the review process.

The phase called “planning the review” has as its purpose to identify the need for this study and to see through the development of a review protocol. A researcher may need a systematic review to be able to draw more general conclusions about a phenomenon or as a prelude to further research activities.

The protocol specifies the methods that will be used to undertake a specific systematic review. A pre-defined protocol is needed to avoid the possibility of researcher bias. Without a protocol, the selection of individual studies might possibly be driven by the expectations of the researcher.

When the whole planning is done, the review can start. This is the second phase, called “conducting the review”. This phase lies in the identification of research, the selection of primary studies, the quality assessment study, data extraction and monitoring, together with data synthesis.

Firstly, the researcher must search the documents by using the strings specified in the protocol. When a first potential set of primary studies is obtained, the researcher must perform a selection by assessing
Table 1: Studies Selection.

<table>
<thead>
<tr>
<th>Author, date</th>
<th>Study name</th>
<th>Source</th>
</tr>
</thead>
</table>

the studies’ actual relevance. Quality assessment must be done over the selected studies. As the result of assessing the information quality, according to the criteria defined in the protocol, a new set of studies is generated.

Finally, the data synthesis provides researchers with the results of the systematic review. The synthesis may be either quantitative or descriptive.

The last phase lies in the communication of the results. Usually the systematic review is reported in at least two formats: In a technical report or in a section of a PhD thesis as well as in a journal or conference paper.

3 REVIEW RESULTS

This section presents the selected works in the searches performed in the digital libraries, journals and internet sites related to the issue in hand. Moreover, a classification of studies is given. This has used aspects which are of relevance to the goal of the review as a basis for this classification.

3.1 Studies Selection

The first step was to search in the predefined information sources. Those sources are: ACM digital library, IEEE digital library, Science Direct Digital Library, Business Source Premier, Wiley InterScience, www.BPTrends.com, www.bpmg.org. The result of this search was a first set, composed of 22 studies. With the aim of tuning the set of studies, the selection criteria were applied. The studies had to contain information about the application of model driven engineering or model driven architecture in business process management.

The issue of the systematic review is MDE for BPM, but because MDA is currently so widespread in the model engineering world, MDA was included in the selection criteria.

As the result of the application of selection criteria, the new set of studies was composed of 10 works (Table 1).

3.2 Classification of Studies

The selected studies have been classified according to several aspects that have been chosen to satisfy the goal of the systematic review (Table 2).

First of all, the author’s opinion about the issue of systematic review is the most important aspect to take into account in classifying the studies. Another important aspect is whether the study offers a proposal about the use of CIM, PIM and PSM (MDA models) within the business process context. This means that the author suggests a specific utilization of MDA models, pointing out the possible modeling standards used in each model. Finally, the different standards proposed by authors for modeling business process are also aspects that are taken into account.
Table 2: Classification of the selected studies.

<table>
<thead>
<tr>
<th>Author, date</th>
<th>MDE for BPM</th>
<th>Propose CIM, PIM, &amp; PSM</th>
<th>UML</th>
<th>BPML</th>
<th>BPMN</th>
<th>BPEL</th>
<th>J2EE</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roser and Bauer (2005)</td>
<td>Yes</td>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>ebXML, AIRIS, WS-CDL</td>
</tr>
<tr>
<td>Zeng, et al. 2005</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pfadenhauer, et al. (2005)</td>
<td>Yes</td>
<td>Partially</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>SBVR</td>
</tr>
<tr>
<td>Rosen (2004)</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Frankel (2005)</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>SBVR</td>
</tr>
<tr>
<td>Harmon (2004)</td>
<td>Yes</td>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>SBVR</td>
</tr>
<tr>
<td>Frankel (2003)</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smith (2003)</td>
<td>No</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kano, et al. (2005)</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>XPDL</td>
</tr>
<tr>
<td>MEGA &amp; Standard Bodies (2004)</td>
<td>Yes</td>
<td>No</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>XPDL</td>
</tr>
</tbody>
</table>

4 FINDINGS AND ANALYSIS

This systematic review goal is to identify studies that can provide an approach for the application of the MDE paradigm to business process management. Note that from here on in the text, MDA will be the modeling approach that will always be mentioned, whereas MDE will not. This is because MDA is the most widely-seen example of MDE application, and because all the papers deal specifically with MDA, and not with MDE in general.

The article “BPM and MDA: Competitors, Alternatives or Complementary” (Smith, 2003), does not share the optimism of the rest of the authors. In Smith’s opinion, BPM and MDA are very different. He declares that MDA must be used by software engineers and that BPM must be used by business people. He also affirms that the latter are not interested in a new approach for developing more software, but rather in a design-driven architecture based on processes and on a business process management system (BPMS) that interprets such designs, in the same way that RDBMS interprets a relational model. Although he does not deny the possibility that in the future the two philosophies may work together, at the moment he advocates the separation of both approaches.

The work “Model-Driven Business Performance Management” (Zeng, et al., 2005) proposes a technical approach for developing a complete application related to the BPM context. This study presents a relation between the two important concepts of this systematic review, using a model-driven approach to build the solution. The technical approach is based on the observation metamodel and its transformations. When the models are transformed, the approach suggests compiling the operational aspects of the model and finally developing a runtime engine that interprets the model and executes the generated code.

The study “Comparison of Two Distinctive Model Driven Web Service Orchestration Proposals” (Pfadenhauer, et al., 2005) focuses on the way to generate a set of web services that implement the organization business processes. By applying the MDA approach, and using some of the business process standards, the final solution is generated. This document mentions the BPDM standard as the MDA BPM connection.

The article “Analysis and simulation of business solutions in a service-oriented architecture” (M. Kano, 2005), offers a four-layer model architecture, in which the first two layers, when viewed together, are similar to the CIM layer in MDA from the business point of view rather than from the software
The last two layers correspond directly to the MDA PIM and PSM layers. By separating the independent platform concerns of a solution from the specific platform concerns and their associated code by means of MDA, the reuse of solution components is supported. Furthermore, the system is more flexible and adaptable to the changes in business requirements.

The work “A Categorization of Collaborative Business Process Modeling Technique” (Roser, et al. 2005), provides a proposal for applying MDA within the collaborative business process framework. Collaborative business processes are performed among different enterprises, which could have different business process development methodologies. Therefore, the creation of a common framework in which the organizations could communicate to each other in terms of business process would be ideal. The authors have spoken about MDA as the common framework for integrating business process from different organizations. They propose to create the business process CIMs, PIMs and PSMs in every organization, by using their own model language for each kind of model. These model languages must be MOF metamodels. Thus, transformations among metamodels can be done. The communication among the enterprises in terms of business process will be done by means of the common CIMs, PIMs and PSMs. These common models are written by using a common metamodel (one for each kind of model) and contain a view for the models of each organization from their CIMs, PIMs and PSMs. Thus, the common framework is well-known for all the organizations.

The two works by Frankel selected in the systematic review, concerning MDA and BPM, (Frankel, 2003 and Frankel, 2005), point to the use of MDA as the methodology that guides business process design, implementation, maintenance and management. Frankel’s theory is that BPM joined to MDA is stronger than BPM alone, and MDA together with BPM is stronger than MDA alone. Moreover, he gives a wide classification of the different business process standards that currently exist. He aims at the aligning of the business process modeling notation (BPMN) with the OMG metamodel BPDM. This would provide portability utility by means of the XMI format and the power of the MDA transformations, in line with the well-known BPMN standard. Although Frankel is optimistic about the application of MDA in BPM, he also warns us about the wide gap that exists between the abstraction represented by a business process model and the specific models that represent the implementation of the business process.

The study “The OMG's Model Driven Architecture and BPM” (Harmon, 2004), has as its goal the use of MDA within the BPM. Harmon puts BPDM at the centre of business process modeling (Figure 2). The rest of business process modeling standards should be transformed directly to BPDM,
even BPMN. He proposes a way to use the different kinds of MDA models (CIM, PIM and PSM) for business process design and implementation. Thus, CIM will be specified in terms of business process by using BPDM; the business rules by means of business rules metamodel (BRM). These models are used by business analysts. PIM are a transformation from previous CIM, specified in a software system metamodel, for example UML. These models are used by software architects. Finally, PSM are built by transforming PIM to the platform specific language in which the business process will be implemented, for example the J2EE UML Profile.

5 CONCLUSIONS AND FUTURE WORK

The systematic review performed provides a complete view of the proposals and opinions existing in the recent literature about MDE paradigm application in business process management.

Most of the works found point to the use of model driven engineering as a valid approach for business process management. There are proposals for the use of MDA in the context of collaborative business process management, where the model driven plays the role of integration standard and allows different organizations to cooperate from a business process point of view. It is also suggested, on the other hand, that MDA is the methodology that drives the organization business process design, implementation, maintenance and management.

Although most authors are in favor of the use of MDE in business process management, there is some rejection of this idea, throwing into relief how far apart both concepts are, and how difficult it is to obtain cooperation to achieve better results.

Business process modeling standards become the key issue for the MDA application in the context of BPM. These standards must be metamodels, which are instances of meta-metamodel MOF. OMG propose the business process definition metamodel (BPDM) as the standard for business process modeling, which has no final version yet (OMG, 2003). BPDM is a semantic description of the logical relations among several elements of any business process description. It is not a notation. Its advantage is that it is a MOF metamodel. Thus, any other notation language, such as BPMN, can be transformed to BPDM. As BPDM is a MOF metamodel, this can be transported via XMI to any business process tool that knows such a metamodel.

The companies only have to define MDA transformations from the BPDM metamodel to executable languages like J2EE or BPEL.

BPMN is the notation standard most frequently used to define business process at a high level. Some authors are quite adamant in their assertions that the next version of BPDM will take on the BPMN standard. Thus, any high level BPMN model will be able to be shared via XMI and transformed to follow the MDA methodology.

In future research, we will monitor the evolution of BPDM and its convergence with the BPMN standard. We will propose a QVT transformation from BPMN to BPDM, as well as from BPDM to a web services metamodel. To do this, the model management framework MOMENT will be used (Boronat, et al. 2005).

ACKNOWLEDGEMENTS

This research has been supported by the project FAMOSO, partially funded by Ministerio de Industria, Turismo y Comercio, FIT-340000-2005-161 Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica 2004-2007 and “Fondo Europeo de Desarrollo Regional (FEDER)”, European Union.

REFERENCES

Pfadenhauer, K., Dustdar, S., Kittl, B., 2005. Comparison of two distinctive model driven web service
orchestration proposals. In CECW’05. Proceedings of the 2005 Seventh IEEE International Conference on E-Commerce Technology Workshops. IEEE.


Short Papers
GENERIC FEATURE MODULES:
TWO-STAGED PROGRAM CUSTOMIZATION

Sven Apel, Martin Kuhlemann and Thomas Leich
Otto-von-Guericke-Universität Magdeburg
P.O. Box 4120, Magdeburg, Germany
Email: {apel, mkuhlema, leich}@iti.cs.uni-magdeburg.de

Keywords: Feature-oriented programming, generics, program customization, software reuse, software product lines.

Abstract: With feature-oriented programming (FOP) and generics programmers have proper means for structuring software so that its elements can be reused and extended. This paper addresses the issue whether both approaches are equivalent. While FOP targets at large-scale building blocks and compositional programming, generics provide fine-grained customization at type-level. We contribute an analysis that reveals the individual capabilities of both approaches with respect to program customization. Therefore, we extract guidelines for programmers in what situations which approach suffices. Furthermore, we present a fully implemented language proposal that integrates FOP and generics in order to combine their strengths. Our approach facilitates two-staged program customization: (1) selecting sets of features; (2) parameterizing features subsequently. This allows a broader spectrum of code reuse to be covered – reflected by proper language level mechanisms. We underpin our proposal by means of a case study.

1 INTRODUCTION

Feature-oriented programming (FOP) aims at feature modularity in software product lines (Batory et al., 2004). Features are increments in program functionality and reflect stakeholder requirements. The key idea of FOP is to map features one-to-one to feature modules. A feature module encapsulates all software artifacts that contribute to a feature in a cohesive unit. FOP targets mainly at large-scale components and compositional programming. Hence, program customization takes place at the level of feature modules, i.e., by selecting and composing a set of desired modules. It is not obvious how this scales down to fine-grained customization needs.

Fine-grained program customization is exactly the aim of an alternative approach, generic and parameterized programming (GPP) (Goguen, 1989; Austern, 1998). The key idea of GPP is to implement program structures as generic as possible and to use these in different contexts by parameterization. This approach is known as very fine-grained since it enables adjusting the types of program elements; but it is not well explored whether GPP is capable for program customization and reuse at a larger scale.

In this paper we examine the differences of FOP and GPP and how do they influence their program customization capabilities. Thereof we derive a set of guidelines in what situations to use which paradigm. Furthermore, we propose a language-driven approach of integrating FOP and GPP to cover a broad spectrum of scales of customization, which we call generic feature modules. Generic feature modules support two-staged program customization: (1) the desired features of a program are selected; (2) the corresponding feature modules are parameterized for fine-grained customization. Besides customizability, this promotes reuse of feature modules and offers potential for reasoning about explicitly represented configuration knowledge in form of parameters.

To underpin our proposal, we present a fully functional compiler on top of FEATUREC++. We use our compiler to apply generic feature modules to a case study.

In this paper we make the following contributions:

- We compare FOP and GPP with respect to program customization; we infer guidelines in what situations which paradigm suffices.
- We propose an integrated language approach that integrates FOP and GPP.
- We contribute a fully functional compiler that implements our language proposal.
- We underpin our proposal by means of a case study.

1http://www.iti.cs.uni-magdeburg.de/iti_db/fcc
2 BACKGROUND

2.1 Feature-Oriented Programming

FOP studies the modularity of features in product lines (Batory et al., 2004). The idea of FOP is to build software (individual programs) by composing features that are first-class entities in design and implementation. Features refine other features incrementally. Hence, the term refinement refers to the changes a feature applies to others. This step-wise refinement leads to conceptually layered software designs.

![Figure 1: Mixin layers.](image)

**Feature modules.** Feature modules implement features. *Mixin layers* is one implementation technique that aims at source code artifacts (Smaragdakis and Batory, 2002). Typically, features are not implemented by single classes; often, a whole set of collaborating classes contributes to a feature. Classes play different roles in different collaborations. A mixin layer is a static component encapsulating fragments of several different classes (roles) so that all fragments are composed consistently. Figure 1 depicts a stack of three mixin layers ($L_1 - L_3$) in top down order. Mixin layers crosscut multiple classes ($C_A - C_C$). White boxes represent mixins and gray boxes feature modules.

**Feature modules in C++.** FEATUREC++ is an extension to C++ that supports FOP (Apel et al., 2005). Feature modules contain classes and refinements to classes, which are declared by refines. Figure 2 depicts a class that implements a simple list (Lines 1-4) and a refinement that determines its size (Lines 5-8). Refinements may introduce new attributes and methods or may extend methods of their parent classes. In our example, the refinement introduces a size field (Line 6), a getSize method (Line 6), and extends the put method by code for item counting (Line 7). To access the extended method the super keyword is used (Line 7). The class List and its refinement belong to two distinct development steps implementing two individual features: a basic list feature (list) and a feature for determining its size (size).

![Figure 2: Refining a list with code for determining the size.](image)

**Figure 2: Refining a list with code for determining the size.**

2.2 Generic and Parameterized Programming

GPP is about generalizing software components so that they can be easily reused in a wide variety of situations (Goguen, 1989; Czarnecki and Eisenecker, 2000). It serves the need for customizing components to specific requirements. In this paper we use C++ templates as a representative approach of GPP (Austern, 1998).

They key idea of GPP is that software components often do not rely on specific data types, but can operate with arbitrary types. In order to reuse these components for all possible kinds of types, they are implemented against type parameters that act as placeholders for different concrete data types. To use a generic component in a specific context, it has to be instantiated by passing a concrete type to the component.

Figure 3 depicts a standard GPP example: a generic list implementation (Lines 1-5). It is generic because it relies on a template parameter $ItemT$ (Line 1). Therefore, it can be used polymorphically in different contexts. When instantiating a list object a programmer has to pass a concrete data type to the list, e.g. $Item$ or $Thread$ (Line 6).

![Figure 3: A generic list and two concrete variants.](image)

**Figure 3: A generic list and two concrete variants.**

### 3 ANALYSIS OF FOP AND GPP

In this section, we analyze the different capabilities of FOP and GPP to implement customizable software by means of an example.

#### 3.1 A Product Line of Linked Lists

As example, we choose a standard problem: a product line of linked lists ($list\ product\ line – LPL$), adopted...
3.2 Implementation of LPL

Ideally, when implementing this product line the programmer implements each feature via one feature module. This is in line with the methodology and principles of FOP. Applying this methodology to our example, we would have to implement at least 11 feature modules, assuming one basic list, three different length counter types, and one item type.

FOP implementation. Figure 5 shows a basic list and a tracing feature implemented in FEATUREREC++. It can be seen that both features are implemented as mixins, encapsulated in feature modules (not shown). Up to here FOP works fine. But implementing other features reveals the weaknesses of FOP. For example, it is not obvious how to implement different variants of the item type feature, i.e., different types of items. One can implement for each type a distinct basic list, of the item type feature, i.e., different types of items. It can be seen that both features are implemented as mixins, encapsulated in feature modules (not shown).

Another approach would be introducing an abstract base class of all items. Hence, the list implementation becomes invariant with respect to the item type. New item types inherit from the abstract base class. This solution imposes performance penalties and a higher resource consumption due to dynamic binding; it may demand for dynamic type checking.

Similar problems occur when implementing other features that come in slightly different variants, e.g., the length counter type, an allocator or iterator type.

GPP implementation. Readers familiar with GPP may notice that the problematic features could be easily implemented using template parameters or alternative mechanisms for generics. Figure 6 depicts a generic list that expects an item type and a type for a length counter, as well as two concrete variants, an item object list with integer counter and a thread list with short integer counter. With GPP, a programmer can define a concrete variant at compile time in a type safe manner. However, one has always to anticipate a potential variation point when implementing a feature. Moreover, it is not obvious how GPP can implement larger program features, e.g., implementing the size feature.

3.3 A Comparison of Fop and Gpp

It seems that both, FOP and GPP, are necessary to implement a highly customizable software. While feature modules encapsulate large-scale features, GPP allows for fine-grained tuning. The questions that arise are what approach is useful under which circumstances? Are both approaches equally expressible? How to integrate both in a consistent way? In this section we shed more light on these issues and provide guidelines for the efficient use of FOP and GPP.
FOP. Feature modules usually contain a set of classes. These classes are introductions or refinements to existing classes. Thus, feature modules implement mainly increments to a program’s functionality. A refinement may extend existing methods by executing code around a method execution. Although refinements rely on a reasonable structure of the base program, they do not expect explicitly represented variation points (e.g., hooks) for being applicable. A feature module binds to the natural structure of the base program and may apply unanticipated changes. A consequence of its encapsulation property is that a feature module can implement a variant that concerns multiple variation points, e.g., a synchronization feature extends simultaneously a list and its iterator.

However, the fact that feature modules rely on given structural abstractions defines the minimal granularity of customization of software built of feature modules. It is not possible to refine a base feature at statement level to change existing types, etc. However, achieving even so customizability at type level (1) imposes performance penalties due to dynamic binding, i.e., implementing a feature against an abstract class, and (2) it results in redundant code, i.e., for each type a distinct feature module. That is, FOP imposes a complexity overhead at small scales.

GPP. GPP supports program customizability at a smaller scale than FOP. For example, templates enable the programmer to customize program structures down at type level by parameterizing types of used variables and arguments. This methodology implies that programmers have to anticipate changes to and variants of a program. Variation points are explicit and fixed; they are an inherent part of the referring modules. n variation points demand for n parameters. This in-language approach to customization facilitates static type-safety.

Although templates can be used to implement entire feature modules (Smaragdakis and Batory, 2002), they are mainly suited for fine-grained customization. This is because the overhead of complexity to maintain the template expressions grows considerably for large-scale features.

Table 1 summarizes our observation of the properties of FOP and GPP achieving customizability and reusability. It is intended to serve as guideline for programmers to decide when to use which technique.

<table>
<thead>
<tr>
<th>Property</th>
<th>FOP</th>
<th>GPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>scale</td>
<td>large</td>
<td>small</td>
</tr>
<tr>
<td>granularity</td>
<td>methods, classes</td>
<td>statements, types</td>
</tr>
<tr>
<td>var. points</td>
<td>implicit</td>
<td>explicit</td>
</tr>
<tr>
<td>extensions</td>
<td>unanticipated</td>
<td>anticipated</td>
</tr>
<tr>
<td>locality</td>
<td>multiple points</td>
<td>single point</td>
</tr>
</tbody>
</table>

Templates. Mixins within feature modules can declare a list of template parameters (Fig. 7). In contrast to traditional classes, subsequently applied refinements to a class may extend its (possibly empty) template parameter list. This is useful because in this way the set of parameterizable types has not to be anticipated up front. Figure 8 depicts a refinement to the basic list that implements the size feature. The type of the counter is passed via template parameter; for that, the template list is extended (Line 1).

Figure 7: A list template.

Figure 8: Extending the parameter list in a refinement.

However, extending the parameter list implies that clients have two provide a set of expected parameters, which may vary depending on the current feature selection. We address this issue later.

Parameterizing feature modules. Templates can be used to parameterize feature modules statically by a set of types. Figure 9 depicts a stack of generic feature modules. Each feature module extends existing structures, but also the template parameter list. This enables the individual features to declare new parameters that are intended for customizing themselves. Composing a concrete program out of this set of features allows the final program to be parameterized with concrete types.

This example illustrates the two-staged nature of the configuration process imposed by generic feature modules: (1) a subset of features is selected; (2) the selected features are parameterized.

4 GENERIC FEATURE MODULES

As our analysis revealed, both, GPP and FOP, have strengths at different scales of customization. Consequently, we propose the notion of generic feature modules that integrates mechanisms of GPP into FOP.
Configuration repositories. Since each refinement may potentially extend the template parameter list, the programmer may easily get lost in the mass of parameters and values (constants and types). In order to simplify the parameterization and to improve code readability without constraining its expressibility and flexibility, we adopt the notion of configuration repositories (Czarnecki and Eisenecker, 2000). A configuration repository encapsulates the overall configuration knowledge that is passed via parameters (Fig. 10).

The key benefit of using configuration repositories in generic feature modules is that now classes and refinements expect only one parameter, namely a repository. Each refinement takes out only this configuration information that it depends on. That solves the problem that each client has to know what set of parameters are expected by the current selection of feature modules. Instead, the configuration repository can be defined in one location; clients do not need to know about its overall structure.

Figure 11 depicts a reimplementation of the basic list and the size feature. Both use different subsets of this repository.

```
template<typename Config> class List {
    typedef Config Config;
    typedef typename Config::ItemT ItemT;
    void put(ItemT *i) { i->next = head; head = i; }
};
```

```
template<typename Config> class ListConfig {
    typedef Config Config;
    typedef typename Config::ItemT ItemT;
    typedef Config::ItemT ItemT;
    typedef typename Config::SizeT SizeT;
    void put(ItemT *i) { super::put(i); size++; }
};
```

Figure 11: Customizing features via repositories.

Discussion. An alternative version of LPL that omits templates could be implemented using abstract classes. Each parameter would be passed as object reference via the list’s constructor. Different parameter settings would be implemented by subclassing abstract classes that serve for representing parameters.

Besides the mentioned penalties imposed by abstract classes, it is not obvious how to bundle parameters in repositories without type definitions and templates. Nevertheless, many design and customization decisions are made upfront. In contrast to using FOP standalone, generic feature modules provide a well-aligned symbiosis between GPP and FOP that supports customizable large-scale components with configuration support and static type safety.

6 RELATED WORK

**GenVoca** is an architectural model for large-scale components and collaboration-based designs (Batory and O’Malley, 1992). Principally, GenVoca distinguishes between horizontal and vertical parameters. The vertical parameters are instrumental in defining the vertical refinement hierarchies of layers, whereas the horizontal parameters provide for variability within a single layer (Goguen, 1996). Mapping this to our approach, concrete configuration repositories *encapsulate* horizontal parameters; vertical parameters are the classes that will be refined. Interestingly, we integrate the configuration repositories into the GenVoca layers themselves, enabling subsequent refinement.

Our notion of configuration repositories builds on an earlier proposal: They are implemented as *trait classes*, i.e., classes that aggregate a set of types and constants to be passed to a template as a parameter (Myers, 1995). Additionally, we provide means...
for integrating configuration repositories into feature modules.

Consul is an integrated approach to manage variabilities and customization (Beuche et al., 2004). It provides a proprietary component model and a logic-based representation of configuration knowledge. The component model lacks the flexibility of mixin composition and compositional reasoning; the logic-based approach of customization is powerful, but relies on a complex program transformation approach. Issues as type-safety are not discussed.

Making configuration knowledge and management explicit is a kind of meta-programming. A comprehensive overview of static meta-programming in C++ is given in (Czarnecki and Eisenecker, 2000). There it is shown how configuration repositories can be further processed to automatically determine parameter settings on the basis of partially specified configurations.

7 CONCLUSION

In this paper, we examined the capabilities of FOP and GPP for implementing reusable software: FOP performs well for implementing composable large-scale building blocks, but it imposes a complexity overhead when implementing fine-grained customizable features; GPP focuses mainly on reuse in the small by providing proper means for fine-grained program customization, but lacks abstraction and composition capabilities for programming in the large. Consequently, we proposed an integrated language-level approach for supporting both kinds of customization and reuse. Generic feature modules impose a two-staged program customization: After selecting and composing features, they can be parameterized to adapt them to a specific application context. A distinguishing feature of our approach is that we integrate the configuration knowledge into the associated feature modules to improve the encapsulation properties. For implementation, C++ templates are only a first attempt to demonstrate our ideas. Exploring sophisticated mechanisms for representing and reasoning about configuration knowledge is part of further work.

ACKNOWLEDGEMENTS

We thank Don Batory and Christian Kästner for useful comments on earlier drafts of this paper. This research is sponsored in parts by the German Research Foundation (DFG), project number SA 465/31-1, as well as by the German Academic Exchange Service (DAAD), PKZ D/05/44809.

REFERENCES


A FRAMEWORK FOR THE DEVELOPMENT OF MONITORING SYSTEMS SOFTWARE

I. Martínez-Marchena, L. Mora-López
Dpto.Lenguajes y C.Computación
E.T.S.I.Informática. Univ. Málaga. Campus de Teatinos. 29071 Málaga, Spain
Email: {ildef,illanos}@lcc.uma.es

M. Sidrach de Cardona
Dpto.Física Aplicada II
E.T.S.I.Informática. Univ. Málaga. Campus de Teatinos. 29071 Málaga, Spain
Email: mсидrach@citima.uma.es

Keywords: OPC technology, monitoring systems, software engineering.

Abstract: This paper describes a framework for the development of software for monitoring installations. Usually, the monitoring of systems is carried out by building a programme for each installation, with no use of previously developed programmes or, alternatively, it is carried out by using SCADA programmes (Supervisory Control And Data Acquisition), although these tools are basically for controlling, rather than for monitoring; moreover, taking into account the small complexity of these type of installations, the use of a SCADA program is not justified. The proposed framework solves the monitorization of an installation in an easy way. In this framework the generation of a monitoring programme consists of three well established steps. The first step is to model the system or installation using a set of generic description rules and the XML language. The second step is to describe the communications among the different devices. To do this, we have used the OPC technology (OLE for process control). With this OPC technology, we have established an abstraction layer that makes it possible to communicate any devices in a generic way. We have built an OPC server for each device that does not depend on the type of device. In the third step, it is defined the way in which the monitored data will be stored and displayed. The framework also incorporates modules that make it possible to store and visualize all the data obtained from the different devices. We have used the proposed framework to build complete applications for monitoring three different solar energy installations.

1 INTRODUCTION

The monitoring of systems is usually carried out by developing a program for each installation or system, especially if the system is not too large or complex. The main purpose of a monitoring process is to reveal the performance of the system, though sometimes the monitoring process is also expected to make a long-term evaluation of the system. However, there is no general framework for the development of systems of monitoring for this type of installations. As a consequence, the generation of a monitoring program does not use previously developed programmes, and usually starts from zero. Moreover, the manufacturers of hardware are unable to make efficient drivers usable for different clients, chiefly because of the differences among their client’s protocols. The tools developed in software engineering for monitoring systems have experimented a huge growth. Nevertheless, these tools usually have no possibility of connecting different systems and applications. It is important to have components that make connectivity easier, (Will et al., 2001), (Feldmann, 2001). That is, the lack of application-level interface standards makes it difficult to interconnect the different applications. To meet the need to distribute object-oriented functionality and encapsulate it with well-defined application programming interfaces, it is developed object distribution models, (Raptis et al., 2001).

We propose the use of the OPC technology (OLE for Process Control) to solve the problem of interconnection. The OPC is based on the OLE/COM technology (Object Linking and Embedding/ Component Model from Microsoft), (Schellenberg, 2001), (Liu, 2005). This technology makes it possible that software components developed by experts in one sector are used by applications in any other sector. The design of OPC interfaces supports distributed architectures. The access to remote OPC servers is made by using the Distributed Component Object Model.
(DCOM) technology from Microsoft, (Horstmann, 1997).

In the following sections a framework is proposed for this purpose, and three phases for developing a monitoring program are described. Finally, the use of the framework for monitoring a photovoltaic solar energy installation is presented.

2 THE FRAMEWORK

In order to develop a program for monitoring a system, the following three steps are required: i) to model the system; ii) to solve the communications among the different components of the system; iii) to define and build a method to store and recover the historical data. Additionally, an easy way to display the information must also be provided. Once these problems have been solved, it is possible to obtain a software for monitoring any installation.

In figure 1 the general scheme of the proposed framework and the elements that make up the system are shown.

3 MODELLING THE SYSTEM

The first step is to model the system or installation using a set of generic description rules and the XML language (XML, www) to describe the elements that make up the system and to implement interoperation between different object distribution models.

The descriptive powerful of XML allows us to extend the possibilities of modelling any system if more parameters of it are to be described. In this way, we can describe a complex system by using easy rules that will have different characteristics and devices in each case.

3.1 The Measure

The minimum unit usable by the system is the measure. A measure is only a representation of one channel from any device that supplies any type of information about the system. A measure can also be one attribute of one device (that we will treat as a constant or calculated value). In figure 2 the scheme used for modelling a measure is shown.

As this figure shows, a measure has several attributes, such as the name, a description, the associated data and the minimum or maximum value.

3.2 The Device

A device is any physical element in the installation. Generally, the devices have some measures and attributes like the name, device class and others. A device can also consist of a set of devices, each of them with its own attributes. In figure 3 it is shown the modelling of a device. A device will be modelled as an element with several measures that can have at the same time a list of elements. In this way it is possible to model a device that consists of several devices and
it is even possible to create one abstract element from a set of devices; in this abstract element its measures could be the calculated value from any of its channels. It is easy to build a data base of the most used devices because it is possible to model a device by using XML labels. In this way, modelling an installation simplifies to selecting the devices that integrate the installation and to assign values to its attributes. Sometimes it is possible that one device consists of several smaller devices that have to be modelled. This can be done by grouping these devices in a set of devices and by assigning to them the necessary attributes.

3.3 The System

The system represents an installation with its devices, its attributes and the associated channels. This is the only information that the framework needs. From the specifications included in the XML document, the framework generates in runtime a structure that connects to the installation and updates the measures with the current data of the different devices. When the framework is started the communications are begun, and at this moment there is a one-to-one mapping between each measure attribute and the corresponding value of the real device channel. In this way, we have always a representation of all the device channels of the installation.

4 THE COMMUNICATIONS DILEMMA AND THE OPC SOLUTION

The second step to build a monitoring program is to describe the communications among the different devices. To do this, we have used the OPC technology (OLE for process control). In any installation it is common to find many devices of different types and manufacturers that have different ways of communication. In order to obtain a generic system we will have to use a general mechanism to communicate with any device, irrespective of their characteristics or the manufacturer. That is, we must give an answer to the question of how to communicate with any device without modifying the system. The OPC (OPC Foundation) technology allows us to solve this question. With the OPC, we have established an abstraction layer that makes it possible to communicate any devices in a generic way. We have built an OPC server for each device that does not depend on the type of device.

The access to remote OPC servers is made by using the Distributed Component Object Model (DCOM), DCOM extends Microsoft’s object-oriented Component Object Model to promote interoperation of software objects in a distributed-heterogeneous environment, (Several, 1995), (Horstmann, 1997). A DCOM server is a body of code that serves up particular object types at runtime. A DCOM client calls into a DCOM server’s exposed methods by acquiring a pointer to one of the server object’s interface.

Once we have modelled the system we must associate the physical channel that will supply the values to each virtual channel. To do this, in the XML specification we add one link that indicates the OPC channel that supplies the value for the virtual channel. In this way we connect the values of the physical devices with our model. For example:

```
<measure code="MeasureCode"
ToDB="true" DBUpdateRate="0">
  <function>
    self.value:=
    :=OPCServer.Group.OPCItem.value;
  </function>
  ...
</measure>
```

Sometimes it is necessary to transform a measure from a device by using a certain function; it is possible to do this in an easy way with the proposed scheme. It is also possible to associate several OPC channels with a measure of the system. For example:

```
<measure code="MeasureCode"
ToDB="true" DBUpdateRate="0"> ...
  <function>
    self.value:=
    :=OPCServer.Group.OPCItem.value+
    +(OPCServer2.Group.OPCItem.value+4);
  </function>
  ...
</measure>
```

Hence we can build any kind of measure as a function of physical channels and numerical transformations.
5 PROVIDING PERSISTENCE

In the third step of our framework, it is defined the way in which the monitored data will be stored and displayed.

Usually it is desirable to have a mechanism to recover and store information. For this reason we will store in a database the parameters and channels that are being monitored. For the database we have used the database manager Firebird (Firebird, www). Firebird is a high-performance and Open Source Database system that can be deployed under several Linux environments.

By using a database system it is possible to provide persistence to the system by just indicating in the XML description what information must be stored and when it must be stored.

For instance, if we want to store the temperature value for 500 ms intervals in the database we have the following sentence:

```xml
<measure code="temperature" ToDB="true" DBUpdateRate="500 ms"/>
```

In this way we will have in a database all the values and states by which an installation has passed throughout all the monitoring time.

6 THE VISUAL LIBRARY

In the previous sections it has been described how to model a system, how to operate the different devices and how to provide persistence to the system. However, in many cases it would be better to have a visual representation of the performance of the system in real time.

The framework that has been developed has several interfaces that allow us outside access using elements that are not the inside elements of the system. For example, it is possible to have visualization components that can be connected to a measure whose value is continuously shown or to graphics with information about the historical values of a channel.

Due to the magnitude of this project and to the large number of options, the outside access has been modelled using design patterns, (Gamma et al., 1995). Specifically, the Observer pattern has been intensively used. In this way, the components only have to select the different elements and these elements will assume the responsibility for notifying the changes, so that the components do the appropriate actions. In this way, we have an easy mechanism for the visual integration of the elements for the monitoring. This mechanism can be easily extended to new components.

7 EXAMPLE: MONITORING A PHOTOVOLTAIC SOLAR ENERGY INSTALLATION. IMPLEMENTATION AND RESULTS

Nowadays this framework is being used for the monitoring of many photovoltaic installations. In these installations each device has several channels and in most cases the devices belonging to different manufacturers, what sometimes makes it impossible to integrate the software supplied by them.

In a photovoltaic installation the following devices are available:

1. Photovoltaic modules that collect the solar energy
2. One or more inverters that transform the direct current collected by the modules in alternating current
3. Several additional devices or sensors that are responsible for collecting the parameters of the performance of the system, such as the temperature, the radiation, etc. In many cases the inverters have
input ports for connecting the sensors and for managing their values.

Not only do the inverters have all the necessary electronic to convert the DC in AC, but also a communication port with its own protocol. This protocol is different for different inverters.

By using a XML format file we will describe all the element in an installation. The modelling of some of the most used devices is already included in a XML labels data base. Therefore, the building of the final system is done by copying the labels of the system devices and by adding them to the original XML. So, for each inverter we will describe its attributes, its channels, the necessary virtual channel and the relationship among them. Finally, we will link the different elements with the OPC items that will be responsible for the communications and the framework that will be responsible for all other tasks. In figure 4 the process of building the monitoring program is shown.

Once the installation has been modelled, we will include in the system all the graphic elements, such as labels, charts, shapes, and so on, and we will connect the channels. In this way we have designed and monitored this system in a fast and easy way. In figure 5 it is shown the final program when it is running.

8 CONCLUSIONS

In this paper we have described the framework that has been developed for building monitoring programs in an easy way. The different tools that are developed are re usable. A library of visual functions, XML and the proposed framework are used for generating programmes that are being used in several real installations. The framework is responsible for the communications.

In a SCADA system, both the design and the logic of the programme are strongly joined and in most cases the design is not very attractive. Modelling a system using XML allows us to make an easy modification and/or extension. Moreover, we obtain an easy monitoring system because the framework is responsible for maintaining a representation of the installation state. It is possible to get a more attractive design because this abstraction allows us to design the visual part by using the classical MFC, VCL or even Flash library.

The proposed framework solves the monitoring of an installation by reusing previously developed components. In this framework the generation of a monitoring programme is done in three phases. One of the main contributions of the framework is the use of XML for modelling the installation and the use of the
OPC technology for describing the communications among the different devices.

The proposed framework has already been used for monitoring several installations.

REFERENCES


The OPC Foundation site: [http://www.opcfoundation.org/](http://www.opcfoundation.org/)


Keywords: Tacit knowledge, Requirements, Tracing, Latent Semantic Analysis, Natural language processing.

Abstract: Pre-requirements specification tracing concerns the identification and maintenance of relationships between requirements and the knowledge and information used by analysts to inform the requirements' formulation. However, such tracing is often not performed as it is a time-consuming process. This paper presents a tool for retrospectively identifying pre-requirements traces by working backwards from requirements to the documented records of the elicitation process such as interview transcripts or ethnographic reports. We present a preliminary evaluation of our tool's performance using a case study. One of the key goals of our work is to identify requirements that have weak relationships with the source material. There are many possible reasons for this, but one is that they embody tacit knowledge. Although we do not investigate the nature of tacit knowledge in RE we believe that even helping to identify the probable presence of tacit knowledge is useful. This is particularly true for circumstances when requirements' sources need to be understood during, for example, the handling of change requests.

1 INTRODUCTION

Requirements specifications are incapable of representing a problem domain in its entirety in all but the most trivial cases. One of the reasons for this is that much of the knowledge about the problem domain is tacit in nature.

The notion of tacit knowledge was first extensively explored by Michael Polanyi in his seminal book “The Tacit Dimension” (Polanyi, 1983). Polanyi briefly summarises tacit knowledge as “knowing more than you can tell”, that is, knowledge that is so inbuilt within your own understanding of a process that awareness of this knowledge is neither apparent, nor explicable. Kevin Ryan (Ryan, 1993) presented a modern corollary when expressing concerns about the role of Natural Language Processing (NLP) in the requirements engineering process. Ryan’s statement that “neither informal speech nor natural language text is capable of expressing unambiguously the myriad facts and behaviours that are included in large scale systems” reflects the tacit knowledge embedded within the problem domain.

Requirements often embody tacit knowledge that the analyst already has, or has uncovered from their analysis of the problem domain. The starting point for our research is that the identification of knowledge would help in two ways. Firstly, it would help the validation of requirements. Secondly, it would help in situations such as system evolution or dealing with requirement change requests, where the provenance of requirements needs to be understood. We are investigating this problem by developing tool support for a form of pre-requirements tracing designed to establish backwards traces from requirements into extant textual source material such as interview transcripts. We hypothesise that where provenance cannot be established between requirements and source material, this may indicate the influence of tacit information during synthesis of the requirements. Of course, there are other reasons for why requirements might lack identifiable provenance, but identifying a lack of provenance is interesting in itself as it permits requirements analysts to determine common sources of requirements ambiguity. This paper explains our approach to pre-requirements tracing and tacit knowledge identification and presents initial results from applying our tool.
2 TRACING AND TACIT KNOWLEDGE

Gotel and Finkelstein (Gotel and Finkelstein, 1994) identify both the need for and the difficulties associated with requirements tracing. They divide tracing into two classes: pre- and post- requirement specification tracing, which are analogous to high-end and low-end tracing as mentioned in (Ramesh and Jarke, 2001). Pre-requirement specification tracing is concerned with the requirement’s life before it is included in the requirements specification. Post-requirements specification tracing deals with life after inclusion. Pre-requirement specification tracing is underdeveloped compared to post-requirement specification tracing. One problem standing in the way of pre-requirements specification tracing is that requirements synthesis often involves much more than a simple transformation process in which information elicited from stakeholders is re-written.

This is particularly well illustrated by the use of contextual elicitation techniques such as ethnographic analysis. Contextual techniques result in a rich description of the problem domain. On the one hand, this makes identification of tacit knowledge easier by the analyst. However, even where a requirement is derived from explicit elicited information with minimal application of tacit knowledge, the relationship between the raw elicited material and the requirement may be hard to identify without careful reading of both. Certainly, the lexical similarities between the source material and the requirement may be very weak.

The impact of tacit knowledge makes the identification of a requirement’s provenance much harder still. A previous study on the use of ethnography in systems engineering (Bentley et al., 1992) analysed the working practices of Air Traffic Controllers (ATC). Embedded within this poorly structured information are examples of tacit knowledge. When confronted with a slow aeroplane about to enter a busy sector in which all flight levels (permitted altitudes of flight) will shortly be filled, the sector chief rerouted the slow aeroplane to another sector as shown in Figure 1.

The ethnographer explicitly identified this as an example of tacit knowledge as at no point are any details about the aircraft in question mentioned, not even the originating sector, yet the chief is still able to reroute the aircraft. When questioned later the chief replied that he knew which aircraft was in question just by looking at the radar. Plausibly, therefore, an analyst experienced in the ATC domain might synthesise a requirement about the radar display that provided the information used by the chief. Since the nature of this information is only implicit in the ethnographic report, the provenance of the radar display requirement would be difficult to trace were the requirement and ethnographic report the only information available for seeking the trace. Dealing with this limited, textual information is the subject of the next section.

3 IDENTIFYING TRACES IN NATURAL LANGUAGE

Requirements are typically represented in natural language. Determining any semantic meaning from natural language will require an understanding of the language that comprises it. Rule based approaches to linguistics are brittle in the face of linguistic variability and do not scale well to new problem domains which introduce unique vocabulary. Alternative approaches rely on statistical properties of the text, this gave rise to the notion that language is understandable by observation, rather than the classical theoretical linguistic approach. Statistical analysis takes place on a body of language, or corpus, and is composed of examples of natural language potentially in the scale of millions of words.

The applicability of corpus linguistics to document processing in requirements engineering has been shown in several problem domains and at different levels. Rolland and Proix provide a general background for the applicability of natural language, and therefore natural language processing, to requirements engineering (Rolland and Proix, 1992). Gervasi and Nuseibeh use automated lightweight techniques to provide automated validation of requirements in some of NASA’s requirements specifications (Gervasi and Nuseibeh, 2002). Sawyer et al. (Sawyer et al., 2005) provide evidence that probabilistic natural language processing is applicable to requirements...
engineering processes across different domains. One such technique is Latent Semantic Analysis (LSA).

Latent Semantic Analysis (LSA) is a vector space technique that results in the formation of a multi-dimensional, document-word space (Deerwester et al., 1990). It is computationally intensive but allows intelligent document query and retrieval whilst overcoming the traditional problems of polysemy (multiple meanings per word) and synonymy (multiple words that mean the same thing) (Berry et al., 1995; Dumais, 1991). The number of occurrences of each word in a document determines the document’s magnitude in that dimension, thereby determining the position of the document in the space. Similar documents appear to cluster together in the space. This clustering can be heightened by reduction of the space to fewer dimensions by singular value decomposition. Similarity can therefore be determined via a variety of algorithms, such as simple Euclidean distance. LSA is commonly accepted to be a shallow technique that accurately manages to approximate human expectations of linguistic comparison.

A simple document-word space technique, although not LSA, has been used by Johan Natt och Dag et al. (Natt och Dag et al., 2005) to determine linguistic equivalence between two different sources of requirements: market requirements and business requirements. The lexical technique used resulted in more than 50% of correct links between requirements being identified. Further, it was estimated that up to 63% of similar requirements could be identified in this manner. However, this technique is based on lexical similarity measures. It has not been determined if this technique can be used to infer semantic similarities across the wide variety of document types required for pre-requirement specification tracing.

4 PERFORMING PRE-REQUIREMENTS TRACING

By searching for traces between requirements and their respective sources it should be possible to determine requirements that are not firmly derived from source material, thereby reflecting an instance of either:

- Poorly sourced knowledge, that is knowledge which is not clearly defined and should therefore be subject of further investigation
- A form of tacit knowledge, whose presence in the requirements specification demonstrates a description of the external behaviour of a tacit process

Note that we are not seeking to measure requirements completeness. Establishing the absence of requirements that represent information explicit in the source material or (even harder) implicit from tacit knowledge, is outside the scope of this work. The tool implements three distinct phases of analysis:

**Collation** All source documentation and the current version of the requirements specification are prepared here. Several steps are performed, such as collating all the documents into a single logical collection for easier processing, tokenisation, stemming and the removal of syntactic elements of speech. The source material is then split into chunks to enable comparison. As currently implemented, the size and content of chunks are determined by a heuristic boundary detection algorithm (Manning and Schütze, 2000)

**Comparison** The semantic equivalence of chunks is determined by use of LSA. Chunks of source material are then compared against chunks of the requirements specification; the similarities are noted. The application of LSA that we propose requires that the contents of all documents are compared to produce a document similarity matrix. The document similarity matrix contains numbers in the range [1,1], where -1 represents content that is semantically divergent, and 1 represents content that is semantically identical

**Analysis** Candidates of matching chunks are presented to the analyst who may filter the results to increase clarity. Only candidate matches are displayed and it is left to the analyst to finally confirm or deny a candidate match

An overview of these operations is presented in Figure 2.

Figure 2: Identification of sources of requirements. Here chunks t(6) and t(7) are likely to be identified by the system as examples of tacit or poorly sourced knowledge as their source is not known. Note that not all source chunks may contribute to the requirements specification.
5 CASE STUDY

In order to test the validity of our approach LSA was used to trace between a concept of operations for a new system and an ethnographic report of the existing system. The ethnographic reports relate to a UK air traffic control system. The concept of operations was developed by Bentley (Bentley, 1994) for a tool to prototype ATC systems. The ethnographic report was scanned from a printed document using optical character recognition techniques. It contained scanning errors that resulted in spelling and grammatical mistakes that we left uncorrected in order to better approximate real world documents. Neither the concept of operations nor the ethnographic data are as vocabulary rich as the newspaper stories considered earlier. Therefore they were much less computationally expensive to perform LSA on. The full process took under a minute on a desktop machine.

We have not yet conducted a study to determine the effects of varying the size of each document chunk, although a trade off becomes immediately apparent. This is that small chunk sizes (e.g. single sentences) can lead to difficulty in analysts accurately interpreting results as there are too many chunks and relations to concurrently track. Larger chunk sizes abstract a lot of the information and result in an overly granular comparison. We decided to use 5 sentences per chunk for this experiment. This is somewhat arbitrary and future versions will use variable size chunks, so for example, the analyst can investigate individual requirements clauses or steps in a scenario. This chunk size was used on both the concept of operations and the ethnographic report.

5.1 Evaluation

Two measures that can be used to demonstrate that LSA is matching human expectation are recall and precision. In order to calculate these measures, it is first necessary to manually determine the correct links between the concept of operations and the ethnographic reports. The recall and precision may then be calculated as follows:-

1. Compute the similarities between chunks
2. Select a threshold, \( \alpha \) in the range \([-1,1]\]
3. Select a chunk of the concept of operations, \( i \)
4. Manually compare \( i \) to all chunks of the ethnographic report to produce a set of matches, \( r_{\text{man}} \)
5. For chunk \( i \) determine all the chunks of the ethnographic report that have a similarity value greater than \( \alpha \) to produce a set of matches \( r_{\text{lsa}} \)
6. Calculate the recall as

\[
\text{recall} = \frac{|r_{\text{man}} \cap r_{\text{lsa}}|}{|r_{\text{man}}|} \quad (1)
\]

7. Calculate the precision as

\[
\text{precision} = \frac{|r_{\text{man}} \cap r_{\text{lsa}}|}{|r_{\text{lsa}}|} \quad (2)
\]

Essentially, recall can be seen as the percentage of correct associations in the current list with respect to the total number of correct associations, i.e. how many correct associations have been discovered at this point. Precision is the percentage of correct associations with respect to the size of the associations list, i.e. how many of the results are correct. It is therefore expected that the recall of LSA will be high when the threshold is low. By setting the threshold to \(-1\) (the lowest threshold possible) all documents will be included in \( R_{\text{lsa}} \), ensuring total recall. In other words, every chunk in the concept of operations will appear to be derived from every chunk in the ethnographic report. However, this will result in poor precision as the number of incorrect associations in \( R_{\text{lsa}} \) is high. As the threshold tends towards 1 precision should increase as the weak and noisy candidate matches are eliminated.

Figure 3: Recall and precision as a function of threshold.

In order to test that LSA can be used to perform semantic level comparison on these sorts of documents, the associations between 4 of the 25 chunks of the concept of operations were recorded against the 85 chunks of the ethnographic report. These manual associations were then used to plot the recall and precision against threshold, as shown in figure 3. This figure is made from a population sample; corresponding confidence interval plots are presented in figures 4 and 5. These plots show the 95% confidence interval for each sampled point, i.e. the range in which 95% of all members of the population are contained within assuming a normally distributed sample, calculated as \( \bar{x} \pm 1.96\left(\frac{s}{\sqrt{n}}\right) \).

Figure 3 clearly shows that as the minimum threshold of relatedness increases the recall decreases and the precision increases. This provides evidence that LSA is approximating human expectations of semantic equivalence for the documents being considered.
Figure 4: Recall and associated 95% confidence intervals.

Figure 5: Precision and associated 95% confidence intervals.

If LSA was providing the opposite of human expectation we would expect to see the precision drop as a function of threshold. If LSA was producing random results we would expect to see no trend at all in the precision and recall curves.

5.2 Badly Sourced Material

We define any chunk as being badly sourced if it has no relatedness to chunks belonging to other documents for $\alpha > 0.1$. An examination of the chunks of the concept of operations that were poorly sourced fell into two main categories:

1. A detailed description of the semantics of shared user displays. These were requirements invented by Bentley as part of his work on shared displays.
2. Chunks where Bentley has used knowledge from his own field work at the ATC centre and knowledge elicited by him from the ethnographer. Neither type of information were explicitly represented in the ethnographic report.

Other, less significant examples of poorly sourced text were due to us erroneously scanning too much of one of the leading pages in the document that contained the concept of operations, but was unrelated to the concept of operations. LSA correctly identified this material as not being associated with the ethnographic report. The results also include examples of the tool correctly identifying poorly sourced chunks of Bentley’s concept of operations as potentially tacit in nature. One example of this is a chunk of text that contains the lexical term ‘strip’. Strip is a common term in both documents, but despite this the chunk is correctly identified as poorly sourced. The chunk deals primarily with a description of the pragmatics of different views of the airspace, such as a written strip view or a radar view. Similarly, despite many instances of the word ‘radar’ in the ethnographic document no strong link is made with this chunk, as LSA has correctly identified that this chunk is primarily concerned with a concept not covered in the ethnography.

6 LIMITATIONS & FUTURE WORK

Our approach assumes that a significant proportion of requirements are derived relatively directly from elicited problem domain information. If most of the requirements are invented rather than derived the number of candidate matches will be too low for the tool to offer any useful insights into requirements provenance. In addition, there are four factors that constrain the circumstances in which our approach is usable:

**Media** are not necessarily in text form. Video, audio and pictorial sources of information may be used to inform a requirements specification.

**Media availability** reduces the accuracy of the system if not all source media are available. The system is likely to identify many cases of tacit knowledge if the amount of source material is relatively small.

**Inconsistent vocabulary** reduces the accuracy of techniques such as LSA. There is potential to incorporate tools such as WordNet (Miller et al., 1990) to determine lexical similarity via synonym sets.

**Document evolution** may result in new associations appearing and old associations being removed.

Within these constraints we believe that our preliminary results demonstrate the potential of LSA to offer insights into requirements provenance and the influence of tacit knowledge. However, as noted above, we need to provide greater flexibility over chunk size. In particular, chunks must map onto the requirements, use cases, business events, or whatever is the natural unit of traceability in the requirements document under analysis. This will inevitably require some manual pre-processing by the analyst.
We also plan to evaluate LSA against other techniques that may yield similar or better results. In particular, text reuse algorithms used in plagiarism detection technologies may provide meaningful output, such as n-gram overlap (Clough et al., 2002), substring matching via greedy string tiling (Wise, 1996) and sentence alignment (Piao et al., 2002).

7 CONCLUSION

We propose a method of pre-requirements tracing that uses a corpus linguistics technique to achieve semantic-level comparison. By splitting up requirements specifications and the source material from which they were derived into chunks and comparing their semantic similarities, it is possible to determine likely sources for each chunk of the requirements specification. Further, this permits us to identify requirements not firmly derived from the supplied source material. We argue that these requirements represent either poorly sourced knowledge or instances of tacit knowledge embedded in the problem domain or the analyst’s mind. We have demonstrated that LSA, a linguistic technique designed to overcome the problems of polysemy and synonymy, can approximate human expectations of semantic relatedness between chunks of source material and their resulting specification. The source material contains less rich text than found in other domains, such as newspaper articles, but is still able to match human expectation. We plan to show that this technique can be used to identify instances of tacit processes and enable pre-requirements tracing on an on-going software development project to update the student registry system at Lancaster University.

REFERENCES


USING LINGUISTIC PATTERNS FOR IMPROVING REQUIREMENTS SPECIFICATION

Carlos Videira, David Ferreira, Alberto Rodrigues da Silva
INESC-ID, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal
cvideira@inesc-id.pt, davi.d.ferreira@inesc-id.pt, alberto.silva@inesc-id.pt

Keywords: Requirements, Requirements Specification Languages, Linguistic Patterns, Parsing Techniques.

Abstract: The lack of quality results in the development of information systems is certainly a good reason to justify the presentation of new research proposals, especially those that address the most critical areas of that process, such as the requirements specification task. In this paper, we describe how linguistic patterns can be used to improve the quality of requirements specifications, using them as the basis for a new requirements specification language, called ProjectIT-RSL, and how a series of validation mechanisms can be applied to guarantee the consistency and correctness of the written requirements with the syntactic and semantic rules of the language.

1 INTRODUCTION

The requirements specification task is one of the most critical steps in the development of information systems. Not only because it encompasses the initial activities, whose results are critical for the success of the succeeding activities, and of the global project, but because it deals with the identification of the scope of the system to be developed, and the problem to be solved. Several surveys and studies (such as The Chaos Report, available at http://www.standishgroup.com) have emphasized the costs and quality problems that can result from mistakes in the early phases of system development, such as inadequate, inconsistent, incomplete, or ambiguous requirements (Bell, Thayer, 1976).

The requirements concept is one of those IS/IT concepts where there is no standard and widely accepted definition. A classical definition from Kotonya says that a “requirement is a statement about a system service or constraint” (Kotonya, Sommerville, 1998). The number of proposals, both research and practical, has grown in the last decade, but there is still not a universal or most accepted practice. The consequence is the use of different approaches for requirements specification, with different levels of formality; the most adopted solution is still the use of natural language to elaborate requirements specification documents.

This paper describes how the identification of the patterns most frequently used in requirements documents can be used to implement a series of techniques to improve the requirements validation process, using a number of parsing components. Section 2 presents an overview of the ProjectIT research program and describes the main features of a new requirements specification language, called ProjectIT-RSL. Section 3 describes the architecture and the parsing algorithms adopted, section 4 presents related work and section 5 overviews the paper and presents the future work.

2 PROJECTIT-RSL OVERVIEW

As a result of the experience gathered from previous research and practical projects, the Information Systems Group of INESC-ID (http://gsi.inesc-id.pt/), started an initiative, called ProjectIT (Silva, 2004), whose goal is to provide a complete software development workbench, with support for project management, requirements engineering, and analysis, design and code generation activities (the work presented in this paper is partially funded by the Portuguese Research and Technology Foundation, under project POSI/EIA/57642/2004 - Requirements engineering and model-based approaches in the ProjectIT research program).

ProjectIT-Requirements (Videira, Silva, 2004b and
is the component of the ProjectIT architecture that deals with requirements issues. The main goal of this project is to develop a model for requirements specification, which, by raising their specification rigor, facilitates the reuse and integration with development environments driven by models. One of the results of this project is a new requirements specification language, called ProjectIT-RSL (Videira, Silva, 2004a and 2005).

The definition of ProjectIT-RSL took into consideration the format and structure of the requirements documents of the projects we have developed, which led to the identification of a set of linguistic patterns associated with requirements. From these patterns we determined the main concepts used in requirements specification, how they are structured, organized, and combined into wider scope blocks. We derived a metamodel of the concepts identified, which is also the base of a profile (called XIS - Silva, Lemos, Matias, Costa, 2003), common to all our tools. Based on the patterns identified, we defined the syntax of ProjectIT-RSL, which was tested in a prototype developed using the features provided by Visual Studio .NET and the .NET Framework (Carmo, Videira, Silva, 2005), and is now being supported by an integrated set of tools, called ProjectIT-Studio. An example of the editor of ProjectIT-RSL is shown in Figure 1.

The complete specification of all ProjectIT-RSL rules is beyond the scope of this paper, and can be presented in more detail in (Videira, Ferreira, Silva 2006). ProjectIT-RSL allows the definition of different “application units”: (1) reusable components, which can be specified independently, and integrated in broader systems; (2) complete executable systems, that can “include” some of the previous ones (reusing their functionality); (3) architectural templates and application templates, which allow pattern reuse and instance reuse, respectively. The rules expressed below in EBNF notation abstract the structure of our requirements document.

The sentences of our requirements documents are divided in two groups, declaration and definition sentences; the first ones just give names to concepts, associating them with a specific type (which is what happens in an Operation Declaration) whereas definition sentences detail the features of a concept (such as an Entity Definition).

Our profile identified three base concepts, Entities, Actors and Operations, defined by the following rules, which basically state that the complete specification of a concept can be done by a number of sentences.

For example, the number of rules currently used for validating an entity specification is already very large, as the following EBNF rules, although not complete, show.
Although we want to allow the users of our tools to use natural language, the parsing mechanisms, as well as the integration with code generation tools, imply that we must restrict the terms allowed to a recognizable subset, such as the fixed terms we have seen in the above rules. This set of rules, called the TS rules (Template Substitution rules), which can be defined and changed for a specific project, enables the incremental evolution of these terms, just by adding more rules, or by defining synonyms between words. This approach not only supports different writing styles and natural languages, but also is the base for the definition of domain specific languages. The rules are stored in groups related to the sections they apply, and as such we have specialized business entities, functional requirements and non-functional requirements rules.

3 PROJECT COMPONENTS

As figure 2 shows, the architecture that supports ProjectIT-RSL is composed by a number of different components, from which we must emphasize the roles of three of them: a text editor, two specialized parsers and an inference engine.

3.1 The Text Editor

The text editor, represented by the package PIT-RSL Plug-in, is a plug-in built upon the capabilities of Eclipse.NET (a port we have performed of Eclipse to the .NET platform), with features such as auto-complete, auto-format, warnings and errors annotations (text underline and vertical bars marks), syntax-highlighting and suggestions.

When the user opens a requirements specification document written in ProjectIT-RSL (a .pit file) with the text editor, it performs an initial full parsing of the document’s contents and starts a read-evaluate-print cycle typical of event oriented interfaces such as the one we are using. Upon detection of a document manipulation, the plug-in sets a timeout mechanism that triggers an event after a configurable short time interval and, which reevaluates the whole document again, applying only the parsing algorithms to new or modified requirements. Therefore, this lazy document’s evaluation mechanism avoids repetitive calls to the parsing mechanism while the user is temporarily typing. Consequently, having in mind the Model-View-Controller (MVC) architecture’s analogy, all model dependent views (subscribers) of the parsing result (the RSL model) stay immutable until new relevant changes occur in the document’s contents presented in the text editor (a view-controller component), instead of being constantly refreshed.

3.2 The Parsing Components

The analysis of the requirements sentences is performed by two parsers, the RSL-Structural Parser and the RSL Fuzzy Matching Parser. The first is generated by a set of tools called CSTools (available at http://cis.paisley.ac.uk/crow-cio/) and performs the initial parsing steps, validating the document’s structure. The second is called PIT-RSL Fuzzy Matching Parser and is responsible for processing Natural Language (NL) text to find the optimal parsing tree, by successive testing NL patterns contained in the TS set of rules.

3.3 The Structural Parser

The generation of the Structural Parser is based upon two script files, one that contains all the regular expressions that recognize the tokens specified in the PIT-RSL language definition, and the other that contains all non-terminal and abstract syntax tree nodes specifications.

One of the first steps of the Structural Parser is to break in tokens its input (the requirements document): it parses the file accordingly with the semantic contexts introduced by the SYSTEM and SECTION tokens. This enables the detection and validation of the requirements hierarchical numeration and enforces a predefined sentences’ structure for each of the above scopes. This step was essential for early detection of potential problems and inconsistencies.

For dealing with the nested structural scopes of the requirements specification document, we introduced a context stack mechanism. Additionally, it was necessary to establish an error and warning mechanism to allow the parsing process to continue to run until the end of the requirements document.
file, even in the presence of non-critical errors, introducing this way a certain level of robustness.

The output of the early stage transformations is an abstract syntax tree (AST) which contains an overall improved representation of the original free-form document, where each requirement is contained in a specific section of the nested structural hierarchy and consists of a sequence of words. Subsequent iterations over the produced AST are used to supply information to other internal data structures belonging to other components which are responsible for the next parsing stages. Before starting to examine the requirements’ semantics, the parsing mechanism must first import all the referenced documents/systems present in the section type “Section Imports”. The import mechanism follows a depth-first approach while parsing and loading the documents/systems specified for import but, for safety, it maintains a path trace which avoids endless importing cycles and redundancies.

3.4 The Fuzzy Matching Parser

The second parsing component is called the Fuzzy Matching Parser (FMP), which heuristically analyses the semantics of each requirement through the adherence of the statement’s semantics to its syntactic structure, which is typical of requirements sentences. Initially, this component performs a morphological and syntactic analysis of each word of every requirement statement. This categorization analysis, based on each word definition and on the context in the statement (relationship with adjacent and related words in a sentence or paragraph), is performed by an external component that implements the Brill Tagger’s algorithm for part-of-speech tagging (more information available at http://www.cs.jhu.edu/~brill/code.html), which marks each word by appending a set of grammatical information tags, providing the required words’ classification for further appliance of the FMP algorithm.

To avoid redundant “labelling” and parsing information, we store a list of all distinct words used in the requirements document, and implement a hashing mechanism that assures words’ uniqueness when adding new words to the list. The use of the word list constitutes an efficient way of managing the terms available and, simultaneously, improves memory usage since it is based on the GIF format compression algorithm (Sayood, 1996), where each sentence is converted in a sequence of numbers, each corresponding to a previously tagged word.

For each new requirement addition, the FMP algorithm is called with the requirement statement and a set of specific TS rules, depending on the section where the requirement belongs, providing a more refined parsing. The algorithm attempts to find the optimal parsing tree by recursively trying, until exhaustion, to match the requirements statement information with the TS rule’s templates part and, upon a successive match, substitutes the matched information with the TS rule’s substitution part. This recursive search is guided by a heuristic function, which gives a score value for each TS rule applied. In each step the algorithm iterates over all TS rules and, for each, tries to find if there is a complete match between it and the statement being parsed.

A complete match occurs when for all elements of TS rule template part there is at least one match between that template element and a word of the requirement’s sentence. If a complete match occurs, then for all valid matches, the algorithm applies a substitution operation which replaces the matched text fragment of the current requirement’s statement by the template part of the TS rule under analysis within the algorithm step. During the replacement process, the template’s variables are bound to the corresponding word values of the statement and this new requirement’s statement is further used in the next recursive step. Then, the algorithm determines an overall score for the step with a parameterized heuristic function which is based in the following three aspects: (1) the “match quality”, which includes the relative word positions, eventual words inversions, and number of words discarded; (2) the TS rule template length, number of variables, and number of constants; finally (3) the length of the requirement statement’s fragment that couldn’t be parsed. At the end of the step, the algorithm calls itself in a recursive manner executing the same behaviour until it reaches a terminal case where no more rules can be applied.

At the end, the algorithm verifies if the achieved results demonstrate a minimal level of parsing quality by comparing its score with a minimum threshold value: if the attained score exceeds the threshold, the algorithm returns the best results found – the optimal parsing tree; otherwise it returns the original requirement sentence. Since this algorithm is intrinsically recursive, we have to guarantee that it neither enters in an endless loop, nor repetitively reapplies the same TS rule, which in both cases mean a possible TS rules specification error, and consequently a PIT-RSL linguistic patterns’ constructs problem. To solve this issue we have introduced a Background Thread Variable.
Timeout Mechanism (BTVTM). Its goal is to run this heavy-weight FMP algorithm FMP in the background. It has an associated timer for only letting the algorithm perform its tasks during a previously specified period, thus assuring that the algorithm eventually ends. This strategy also guarantees that the tool always provides feedback to the end user instead of blocking each time the algorithm runs during the parsing process.

3.5 Inference Engine

Finally, and to allow further knowledge inference capabilities, important for requirements validation, PIT-Studio/RSL uses two other components: RSL-to-RDF/OWL and Jena .NET Port. The former contains the adapter pattern code that provides a clean C# API for using the .NET ported Jena framework without the necessary traces of java syntax code. The Jena .NET Port package represents a .NET port of Jena framework (available at http://jena.sourceforge.net/), which supplies the PIT-RSL plug-in with knowledge-base and inference-engine capabilities.

4 RELATED WORK

The use of natural language in the initial phases of the software development process has received attention for more than 20 years. Abbot (Abbot, 1983) proposed that nouns could be used to identify classes, adjectives to identify attributes, and verbs to identify methods. OICSI is a tool developed by Rolland and Proix (Rolland, Proix, 1992), to help the identification of requirements from natural language text and available domain knowledge. Attempto Controlled English (ACE), first described in (Fuchs, Schwitter, 1996), is one of those approaches that use a controlled natural language to write precise specifications that, for example, enable their translation into a first-order logic similar representation (called DRS).

The use of parsing techniques to elaborate a conceptual model from natural language requirements is a common approach; in (Macias, Pulman, 1993) we can find descriptions of proposals to use a controlled natural language with a limited syntax in order to specify requirements with more quality. Some of the previous initiatives were concerned with detecting problems in previously written requirements documents (Fantechi, Gnesi, Lami, Maccari, 2002), while others are concerned with the elaboration of requirements documents without such problems (Ben Achour, 1998, Denger, 2002). NL-OOPS (Mich, Garigliano, 1999) and LIDA (Overmyer, Lavoie, Rambow, 2001) are systems that process natural language requirements to construct the corresponding object-oriented model. A similar system is described in (Nanduri, Rugaber, 1996). Although the number of initiatives seems to justify the potential of natural language requirements, there are studies reporting problems in using natural language requirements specifications (Berry, Kamsties, 2003).

A number of different approaches have researched on the elaboration of requirements specification using patterns of natural language. Approaches such as (Ben Achour, 1998) and (Rolland, Proix, 1992) reduce the level of imprecision in requirements by using a limited number of sentence patterns to specify a requirement for a particular domain. Denger (Denger, 2002) has also identified natural language patterns used to specify functional requirements of embedded systems, from which they developed a requirements statements metamodel. Juristo and Moreno try to formalize the analysis of natural language sentences in order to create precise conceptual model (Juristo, Morant, Moreno, 1999).

Ambriola and Gervasi proposed the CIRCE project (sometimes defined as a “lightweight formal method”) (Ambriola, Gervasi, 2003), which uses natural language as the specification language, and is also supported by fuzzy matching parsing techniques to extract knowledge from requirements documents and produce a formal validation of requirements. Although Circe and ProjectIT-RSL have some similarities, there are between them many differences, namely in the architecture, concepts and algorithms used, and above all, in the strategy: the goal of CIRCE has initially been requirements validation, and only recently integrated with model driven approaches, whereas our goal with requirements specification is to obtain a consistent requirements document that enables the use of model driven techniques and code generation.

5 CONCLUSIONS AND FUTURE WORK

The importance of requirements specification led us to propose a new specification language, closely supported by a number of tools that cover most of the requirements specification process, mainly the specification and validation steps. This paper
focused on the description of the parsing steps algorithms, following others where we have described the language ProjectIT-RSL in more detail (Videira, Ferreira, Silva, 2006). The language and the tools have already reached an important maturity level, and the application in small examples has led us to conclude that, although sharing points with other initiatives, we think that our approach has a unique combination of ideas that has not been tried.

In the near future we will concentrate in the development of the requirements reuse mechanisms and in advancing tool support. For example, we will automate the generation of the TS Rules from the ProjectIT-RSL abstract rules, and we will develop plug-ins to show, in different formats, the information stored in the knowledge base. When our ProjectIT-RSL and its supporting tools reach a sufficient maturity level, it is our intention to use them in real projects, to better test and proof the ideas we are proposing.

REFERENCES

Abbot, R., Program design by informal english description, Communications of the ACM, 16(11), pp. 882-894, 1983
Fuchs, N., Schwitter, R., Attempto Controlled English (ACE), CLAW 96, First International Workshop on Controlled Language Applications, University of Leuven, Belgium, March 1996
Sayood, K., Introduction to Data Compression. Morgan Kaufmann, 1996
Videira, C., Silva, A., A broad vision of ProjectIT-Requirements, a new approach for Requirements Engineering, in Actas da 5ª Conferência da Associação Portuguesa de Sistemas de Informação, Lisboa, Portugal, November 2004c
Videira, C., Silva, A., Patterns and metamodel for a natural-language-based requirements specification language, CiSe 2005 Forum, Porto, June 2005
Videira, C., Silva, A., A linguistic patterns approach for requirements specification language, Euromicro SEAA 2006 Conference, Dubrovnik, August 2006
AN APPLICATION OF THE 5-S ACTIVITY THEORETIC REQUIREMENTS METHOD

Robert B. K. Brown, Peter Hyland, Ian C. Piper
Centre for e-Business Applications Research (CeBAR), University of Wollongong, c/o SITACS Bldg3,
University of Wollongong, NSW 2522 Australia
bobbrown@uow.edu.au, phyland@uow.edu.au, ian@uow.edu.au

Keywords: Requirements Analysis, Software Architectures, User Modelling, Activity Theory.

Abstract: Requirements analysis in highly interactive systems necessarily involves eliciting and analysing informal and complex stakeholder utterances. We investigate if Activity Theory may provide a useful basis for a new method. Preliminary results indicate that Activity Theory may cope well with problems of this kind, and may indeed offer some improvements.

1 INTRODUCTION

One of the most crucial aspects of highly interactive, multi-user, organisational systems is the interface. The Human Computer Interaction (HCI) community has not adopted rigorous Formal Methods with open arms (Paterno, 1996). However, the HCI community has widely adopted Usability Engineering approaches (Corporate Solutions 2006), such as Nielsen’s (1994), which offers considerable formality. There remains, however, scope for user interface (UI) design to adopt a theoretical framework to enhance consistency across the whole design and development lifecycle.

A theoretically-consistent framework from initial conceptual elicitation to evaluation of the finished product may prove useful. Since the aim of UI design is to produce interfaces that assist users to carry out their day-to-day activities, particularly in an organisational setting, a psychological and sociological theory could be a serious candidate for the informing theoretical framework. We suggest that Activity Theory (AT) would be a useful framework and could serve as the basis for an end-to-end system analysis and design method for highly interactive, multi-user systems. In this paper, we present an AT-based analysis and design method (called the 5-S Method) and a preliminary test example, used to test the method and explore the suitability of AT.

2 SCOPE

This research focuses on highly interactive systems (Brown, 2005) where the UI itself underpins a large proportion of the system’s functionality. AT is an appropriate theoretical framework for highly interactive systems for three reasons: 1) it is focussed on understanding real life activities, 2) in its classic formulation it provides a method of task decomposition and 3) in its latest versions it has been used to describe networks of inter-related activities. Because AT provides a mechanism for describing networks of goal-directed human activity, it could be useful in understanding those systems that have many users, with multiple roles, whose activities are highly interrelated e.g. most organisational information systems.

<table>
<thead>
<tr>
<th>AT layer</th>
<th>Doing</th>
<th>Facilitator</th>
<th>Driver</th>
<th>Product</th>
<th>Protagonist</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Activity Network</td>
<td>System</td>
<td>Agenda</td>
<td>Process</td>
<td>Group</td>
</tr>
<tr>
<td>3</td>
<td>Activity</td>
<td>Tool/ScreenSet</td>
<td>Motive/Object</td>
<td>Outcome</td>
<td>Subject</td>
</tr>
<tr>
<td>2</td>
<td>Action</td>
<td>~ Screen</td>
<td>Goal</td>
<td>Transaction</td>
<td>Actor</td>
</tr>
<tr>
<td>1</td>
<td>Operation</td>
<td>Switch</td>
<td>Condition</td>
<td>Change</td>
<td>Operator</td>
</tr>
</tbody>
</table>
So the scope of our research is to develop an AT-based analysis and design method specifically for highly interactive, multi-user, information systems. The concept of an Activity Network and the task decomposition inherent in AT i.e. Activity > Action > Operation, allows the proposed method to focus on many different levels of the interaction process. At the higher levels i.e. Activity Network and individual Activities, the method would support more experienced designers who could draw on their own experience to provide solutions to lower level design issues. At the lower levels i.e. Action and Operation, the method would guide neophyte analysts and designers, even to the selection of suitable widgets. So, while our method is at times highly prescriptive, it is also intrinsically flexible, allowing analysts and designers to select those parts of the method which are appropriate to their level of expertise.

3 ADAPTING AT

AT identifies an Activity as the smallest meaningful task carried out by a human subject. Vygotsky states that all human Activity is carried out by a Subject, using physical or psychological Tools to achieve some Object which may result in a physical Outcome (Vygotsky, 1978).

Engström (1987) expanded the conception to include a social context. Figure 1 shows the seven node Engström matrix.

To adapt AT to a system design role, it is necessary to shift focus to the facilitating Tool(s) of an Activity, as these Tools include the computer system to be specified. Ultimately, the analyst is seeking to identify and describe some common set of Tools, at least part of which resides in the Tool node of each member Activity in the Activity network, thus describing a useful computer system.

Leont’ev (1978) proposed a three layer hierarchic structure: Activity, Action and Operation to represent different levels of intellectual “engagement” of the Subject, with an Activity requiring deep engagement while an Operation is virtually autonomic. Kuutti (1991) included a fourth and topmost abstraction: the Activity Network, being that related cluster of Activities that are carried out by a community of Subjects working on some common task or process.

As described (Brown, 2006), we have extended the AT taxonomy to avoid confusion between the four layers. This extended taxonomy is shown in Table 1. English lacks a common collective noun for the abstract notion of ‘verb’, so we employ an atypical definition of ‘Doing’ in the singular (OED). The collective terms ‘Facilitator’, ‘Driver’, ‘Product’ and ‘Protagonist’ were adopted for other AT aspects.

4 AN AT ANALYSIS AND DESIGN METHOD

The 5-S method elicits and decomposes stakeholder utterances, in accordance with AT principles. Starting at layer 4, the Activity Network is identified, layer 3 then identifies Activities. Layer 2 identifies Goal driven Actions and layer 1 atomic Operations. Conditions which drive Operations are then mapped to Switches, a term we employ generically for UI elements. These are recomposed and grouped into the following UI structures:

1. System: The computer tool(s) which best facilitate the Network of Activities.
2. Station: Activities grouped according to Roles within the stakeholder organisation.
3. ScreenSet: Groups of Screens associated
4. Screen: Interface groupings of Switches closely related to Actions within the Activity.
5. Switch: Unitary elements of the UI.

Careful analysis of the requirements gathered at each layer should permit recomposition of the Facilitators at each layer until ultimately a System (the most abstract Facilitator) is described. The description of the System would, for all practical purposes, form a feasible, defensible and consistent Requirements Specification.

As Figure 2 shows, the 5-S workflow passes downwards through the AT layers from abstract to refined, before passing back up through the layers in recomposition. The boundaries of the seven phases are porous in both dimensions.

Horizontally, there are links between the decompositional analyses at any given layer and the

Figure 1: Engström’s Activity Matrix (Engström, 1987).
guidance they give to the recomposition in the upwards pass. Vertically, each of the phases tends to confirm the results of the previous, and yield candidate solutions to the next.

Starting at the most abstract fourth layer, 5-S elicits clues from stakeholders to the Activities. This requires some degree of iterative consultation akin to Business Process Modelling, which serves to confirm and amalgamate the Stakeholders’ consensus view of the process in hand.

Further details of AT and the informing principles of our method have been presented elsewhere (Brown, 2006).

6 PRELIMINARY RESULTS

To explore the suitability of AT in Requirements Analysis, we have run the method as it exists against the test example described above. We present below an indicative selection of preliminary results in the early phases of the method.

6.1 Phase 1 – Activity Network

An Activity Network is a related set of Activities which contains and describes the hierarchic component Doings of a Group Process. We are interested in the requirements for a System that best Facilitates the Group Process. This System comprises computer based Tools which Facilitate and in some instances Automate the Group Doings. To this end, we are interested in those Activities whose Tools could include some element of the System. The user interfaces of the included computer based Tool(s) define a boundary surface for the conceptual space where the System resides. Activities whose Tool nodes do not connect to this surface are not considered.

During initial elicitation, this System does not yet exist and iterative consultation with the stakeholders is advised prior to automating or altering any Doing. These early phases comprise a Business Process Modelling (BPM) exercise. Interestingly, Martins and Daltrini (Martins 1999) have observed that AT precepts are compatible with Yu’s i* BPM method.

Table 2: Phase 1 Elicitation Questions.

<table>
<thead>
<tr>
<th></th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What is the purpose of the Process? (Agenda – layer 4)</td>
</tr>
<tr>
<td>2</td>
<td>Who is involved in this Group? (Subjects – layer 3)</td>
</tr>
<tr>
<td>3</td>
<td>What classes/roles of people are involved? (Roles – layer 4)</td>
</tr>
<tr>
<td>4</td>
<td>What does this Group do? (Process – layer 4)</td>
</tr>
<tr>
<td>5</td>
<td>What do each of these people/classes/roles intend to achieve? (Object – layer 3)</td>
</tr>
<tr>
<td>6</td>
<td>What do each of these people/classes/roles produce? What is their result? (Outcome – layer 3)</td>
</tr>
<tr>
<td>7</td>
<td>Why do each of these people/classes/roles carry out their Activity? (Motive – layer 3)</td>
</tr>
</tbody>
</table>

As the scope of the System remains unknown, in the early Phases of the method, heuristics are required by which to accept or reject Activities from the
Activity Network and its Process before the System can be described. In this informal analysis, commonalities of several forms between Activities are identified. Generally these are neither necessary nor sufficient conditions. Commonalities we specifically examine include:

- **People**: A Subject in one Activity may be the same person as the Subject in another. A Subject of one Activity may be a Community member of another. It is also important to understand Roles played by individuals, subsets of whom have a part-whole relation with the Subjects of identified Activities.

- **Motive/Object**: if several people express the same Motivation or Object, then they are likely to be Subject members of the one Activity. If these are consistent with a group Agenda and contribute to some group Process, then membership of the Activity network may be strongly indicated.

- **Outcome**: The outcome of one Activity may become a Tool or Rule of another. One Activity may determine the Subject of another (Vrazalic, 2004).

We conduct elicitation of these informal diagnostic characteristics using Phase 1 questions shown in Table 2. Actual interviews are somewhat flexible of course, and these questions serve more as a guideline than as any kind of script. Collection and analysis of these Phase 1 indicators necessarily generates a list of strong candidate Activities, to be confirmed in Phase 2.

Our preliminary results include:

- **Agenda**: Students must demonstrate their learning and skills by completing indicative assessment tasks to a measurable standard without cheating.

- **Subjects**: S1 Academic; S2 Student(s); and S3 Tutor(s). If Subjects are in a part-whole relationship (eg: some differences between an Activity conducted by a single Tutor, or by the Group of Tutors), there are three likely consequences:
  - Firstly, if the Doing of the Subject subset can be conducted in the absence of the rest of the Subject group, then the Actions within the Activity must be designed to allow for some or all or the Subject(s) to conduct the Doings individually as required.
  - Secondly, If the Doing of the subset must be conducted in the absence of the rest of the Subject group, then the Activity needs to be split into two or more, one in which the Activity is conducted by the entire Subject group, other(s) conducted by some subset of the group.
  - Or finally, it may be necessary to create an entirely new Subject, consisting of some part-whole subset of the previous Subject group (and possibly others), essentially a new Role, for this Activity and/or related Activities.

Roles: Subject Co-Ordinator, the highest appeal, records grades etc; Expert Authority, who set assessment, define questions, define answers; Head Tutor, a possible liaison between lower grade tutors and the Academic; Normal Tutor, who distributes, collects, possibly marks and reports; Low-Grade Tutor, who only distributes and collects, no marking; and Student, who must complete assessment on time, without cheating.

### Table 3: Candidate Academic Activities

<table>
<thead>
<tr>
<th>Subject 1A (S1A) = Academic A</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S1A.01 Create assessment questions</td>
<td></td>
</tr>
<tr>
<td>S1A.02 Post assessment questions to FTP site</td>
<td></td>
</tr>
<tr>
<td>S1A.03 Post assessment questions and marking guide to Tutors</td>
<td></td>
</tr>
<tr>
<td>S1A.04 Field clarifications from Tutors and Students</td>
<td></td>
</tr>
<tr>
<td>S1A.05 Pre-process combined answers from all submitting students</td>
<td></td>
</tr>
<tr>
<td>S1A.06 Facilitate negotiation of marking scheme with Tutors</td>
<td></td>
</tr>
<tr>
<td>S1A.07 Conflate all marks from Tutor(s) to a Spreadsheet</td>
<td></td>
</tr>
<tr>
<td>S1A.08 Anonymize Spreadsheet to PDF document</td>
<td></td>
</tr>
<tr>
<td>S1A.09 Upload PDF to FTP site</td>
<td></td>
</tr>
<tr>
<td>S1A.10 Field student appeals and complaints</td>
<td></td>
</tr>
<tr>
<td>S1A.11 Transfer Spreadsheet totals to new Spreadsheet for personal archiving</td>
<td></td>
</tr>
<tr>
<td>S1A.12 End of semester processing of totals to Campus Administration system.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject 1B (S1B) = Academic B</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>S1B.01 Create assessment questions</td>
<td></td>
</tr>
<tr>
<td>S1B.02 Create marking guide</td>
<td></td>
</tr>
<tr>
<td>S1B.03 Distribute assessment questions and marking guide to Tutor(s)</td>
<td></td>
</tr>
<tr>
<td>S1B.04 Distribute hardcopies of Assessment questions to Students in lecture class</td>
<td></td>
</tr>
<tr>
<td>S1B.05 Field clarifications from Tutor(s) and Students</td>
<td></td>
</tr>
<tr>
<td>S1B.06 Conflate marks from Tutor(s) to personal archive Spreadsheet</td>
<td></td>
</tr>
<tr>
<td>S1B.07 Upload marks to central Website</td>
<td></td>
</tr>
<tr>
<td>S1B.08 Field Student appeals and complaints</td>
<td></td>
</tr>
<tr>
<td>S1B.09 End of semester processing of totals to Campus Administration system.</td>
<td></td>
</tr>
</tbody>
</table>

Identification of people, motives and outcomes informed the choice of individuals to be interviewed. Interviews with Academics A and B, and some Students and Tutors of each produced candidate Activities. Different individuals expressed different personal interpretations of the process, which
resulted in multiple sets of responses. Table 3 shows the candidate Activities elicited from the Academics. The System is being designed to facilitate the Group Process and so iterative elicitation of stakeholders is required to reduce these two sets to a single consistent list of Activities. This occurs in Phase 2. Except in so far as their effects are reflected in the Division of Labour, description of Roles plays no direct part in AT, and as such the term does not appear in our extended AT taxonomy. Whilst in this example eliciting Roles proved necessary for a consistent Activity list, it may not always be necessary to elicit Roles simply to move on to Phase 2. We anticipate however, that a clear mapping of the coincidence of Roles in Subjects will prove necessary for recomposition into System Requirements in Phase 7. Further, there may be times during decomposition of Group Doings that an analyst is tempted to restructure the Group Process by collapsing or conflating Doings. Consideration of Roles however should reveal that some near-equivalent and seemingly repetitive or redundant Doings probably must be retained for reasons of the Agenda and the Group’s cultural-historical structure.

6.2 Phase 2 – Activities

The primary unit of analysis in AT is the Activity itself, usually visualised as the seven node Engström matrix (Figure 1). For our purposes we specify what each node contains for a systems design context.

SUBJ: the Subject is the group or individual who conducts a particular Activity. An individual can be the Subject of any number of Activities, which indicates that individuals’ unique organisational Role.

COMM: the Community comprises all other parties involved in transactions associated with the Activity. Subjects of one Activity are often Community members of another, as the Network of Activities at layer four indicates that the Activities are related, if only by use of some common Tool.

Identification of the Community indicates how the UI elements of the System need to be grouped, such that all parties have the appropriate capability to interact and conduct their normal transactions.

TOOL: the Tool(s) comprise all physical, virtual and psychological facilitating mechanisms used by the Subject and or Community members. Tools include not just sophisticated artefacts and softwares, but also seemingly mundane facilitators such as clocks, telephones, notepads and personal conversation. The analyst must consider all Tools, as the final System may subsume, imitate or compliment any number of them.

RULE: the Rules node for our purposes contains primarily Temporal constraints, including ordinal ranking of Actions where appropriate.

DivLAB: the Division of Labour for our purposes contains primarily Deontic constraints. Issues of obligation, permission, denial and the like, indicate ‘who does what’.

OBJ: the Object is crucial analytical node for classical AT, but for our purposes may be conflated with the driving Motive. It effectively contains that which the Activity hopes to achieve.

The identification of Objects in early phases of elicitation serves as strong indicators that Activities have been identified. Ultimately, Activities are defined and differentiated by these Motives.

OUT: The Outcome node contains that which actually results from the Activity. Classical AT pays strong attention to the tension between Object and Outcome. For our purposes the Outcome can contain interesting linkages. As observed by Kuutti (Kuutti, 1991), the outcome of one Activity may appear in any node of another: in RULE as a Temporal constraint, in DivLAB as a Deontic constraint, in TOOL as some process, device or document, in SUBJ or COMM as some individual whose Role has changed or most interestingly, in OBJ as a new
motive which thus can instantiate a whole new Activity.
In our test example we elicited different candidate Activities from two Academics, their Tutors and Students. Since the Role(s) played by different members of S1 are equivalent, we assume that there should be a consistent common set of S1 Activities. We returned to the individuals and produced a consensus Activity set which is shown in Table 4. Achieving consensual agreement is not a deterministic process however, though by following AT principles it was possible to facilitate negotiations by presenting all options within a common framework. Some Activities have been subsumed into others, for example S1A.06 “facilitate negotiation of marking scheme with tutors” becomes simply one means of achieving S1.02 “create marking guide”. Other Activities may have been relegated to the Action layer. Should it prove impossible to produce a consensus Activity set, then perhaps there has been some confusion regarding Roles. The Activity Network needs re-examination and perhaps a new category of Subject(s) is required. Thus, following AT principles prompts and facilitates resolution whenever analysis fails to capture the Group Process properly.

By a similar process a consensus Activity set was produced for the Students and Tutors. We present the Student Activities in Table 5.

### 6.3 Phase 3 – Actions

Actions are Goal driven Doings, subsidiary to Activities. The Actions comprising any one Activity should all serve the Motive of that Activity, just as Activities of one Subject must fulfil that Subjects Role. Examining Activity S2.01 ‘get assessment questions’ we identified Goal driven component Actions. These are identified in Table 7.

<table>
<thead>
<tr>
<th>Subject 2 (S2) = Student (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2.01.01 get notification of assessment questions availability</td>
</tr>
<tr>
<td>S2.01.02 get assessment questions</td>
</tr>
<tr>
<td>S2.01.03 get supporting materials</td>
</tr>
</tbody>
</table>

Observe that Action S2.01.02 ‘get assessment questions’ has the same name as the parent Activity S2.01. Despite appearances, this is not an inconsistency, as AT tracks the protagonists cognitive involvement. Actions are of a lower order than their parent Activity. The analyst must however, be careful in situations of this kind and keep the nomenclature convention in mind. Wherever possible, it is better to describe Activities by their Motives, and Actions by their Goals.

### 6.4 Phase 4 – Operations

As Phase 4 marks the turnaround from decomposition to recomposition it attempts to specify a Switch of the proposed System for each Operation that involves the System.
Utterances from Students of Academics A and B, indicated different Conditional Operations, shown in Table 8. One set follows a manual process, the other an online process.

Some confusion arose in negotiating common Operations. The utterances “I go to the right lecture class” and “I go to a networked computer” initially seemed to equate. Asking ‘why’ questions, revealed otherwise. Going to the correct lecture class in fact functionally equates with going to the correct course sub directory after logging on to the network.

Table 8: Candidate Operations for Action S2.01.02.

<table>
<thead>
<tr>
<th>Subject 2A (S2A) = Student of Academic A</th>
<th>Subject 2B (S2B) = Student of Academic B</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2A.01.02.01 go to networked computer</td>
<td>S2B.01.02.01 go to correct lecture theatre</td>
</tr>
<tr>
<td>S2A.01.02.02 logon to FTP network</td>
<td>S2B.01.02.02 collect assessment questions</td>
</tr>
<tr>
<td>S2A.01.02.03 go to appropriate sub directory for the correct course</td>
<td>S2B.01.02.03 check assessment question document for completeness and correctness</td>
</tr>
<tr>
<td>S2A.01.02.04 go to the appropriate sub directory for the correct assessment task</td>
<td>S2B.01.02.04 ask clarifying questions regarding the assessment questions and/or the constraints they impose</td>
</tr>
<tr>
<td>S2A.01.02.05 download, copy or print out the assessment questions</td>
<td>S2B.01.02.05 leave</td>
</tr>
<tr>
<td>S2A.01.02.06 check assessment question document for completeness and correctness</td>
<td></td>
</tr>
<tr>
<td>S2A.01.02.07 logoff</td>
<td></td>
</tr>
</tbody>
</table>

While some Operations are subsumed, dropped or added, others were outside the scope of the System. Operation S2.01.01 had no initial equivalent for S2B however, Academic B decided that it was a useful feature, and agreed to impose this Condition. Operation S2B.01.02.04 was removed and migrated to Activity S2.04, now expanded and associated with all Student-to-Academic/Tutor communications. Common Operations are shown in Table 9.

Even after the System is deployed, not all Doings will invoke its use. Numerous technical, psychological and mechanical Tools are available. By our definition however, at least some Doings of each member Activity will invoke the System. Our aim is to capture and describe these as System Requirements.

Table 9: Operations for Action S2.01.02.

<table>
<thead>
<tr>
<th>Subject 2 (S2) = Student (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2.01.02.01 establish identity</td>
</tr>
<tr>
<td>S2.01.02.02 select correct course</td>
</tr>
<tr>
<td>S2.01.02.03 select correct assessment</td>
</tr>
<tr>
<td>S2.01.02.04 download/Copy/Print assessment</td>
</tr>
<tr>
<td>S2.01.02.05 verify assessment</td>
</tr>
<tr>
<td>S2.01.02.06 leave</td>
</tr>
</tbody>
</table>

Table 10 shows Phase 4 identified Conditions and maps them to UI widget Switches. Operation S2.01.02.01 can be achieved by a System Logon Doing. For our purposes, complex multi-part GUI widgets such as a FileSave dialogue, are deemed atomic by their near universal adoption.

Operation S2.01.02.03 ‘select correct assessment’ implies exclusive choice from a finite number of pre-set options. Accordingly a Radio Button panel or a Pull-Down Menu may be suitable. Several standard Switches may be suitable and the choice would reflect the personal leanings of the analyst (and/or stakeholders). The analyst should be confident that the design would be functional, appropriate and feasible at least, if not necessarily elegant.

Table 10: Operations and Switches.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2.01.02.01 establish identity to the system</td>
<td>LOGON widget</td>
</tr>
<tr>
<td>S2.01.02.02 select correct course</td>
<td>Radio button or Drop down menu</td>
</tr>
<tr>
<td>S2.01.02.03 select correct assessment</td>
<td>Radio button or Drop down menu</td>
</tr>
<tr>
<td>S2.01.02.04 download, copy or print assessment questions</td>
<td>FileSave and/or Copy-Paste function and/or Print widgets</td>
</tr>
<tr>
<td>S2.01.02.05 verify correct assessment questions</td>
<td>Task switch to local system</td>
</tr>
<tr>
<td>S2.01.02.06 leave</td>
<td>LOGOFF widget</td>
</tr>
</tbody>
</table>

6.5 Later Phases - Recomposition

The Switches identified in Phase 4 will be composed into Screens in Phase 5. Screens and Actions are closely related but there might not be a 1:1 mapping. We do however expect that ScreenSets will have a 1:1 mapping to Activities. Leont’ev (Leont’ev, 1978) predicts that familiarity and expertise leads to Doings dropping down the hierarchy. After some experience, we could able to
short-cut some of the more perfunctory mechanisms, as our own Actions became Operations. This interesting confirmation of AT also indicates that the method will ultimately serve both as a prescriptive toolset for the novice and an informing framework for experienced practitioners.

The analyst however, should collect data in decomposition that serve for the recomposition phases (see Figure 1). Roles help inform Phase 7; the Motives of Activities help compose ScreenSets in Phase 6. Temporal and Deontic constraints, recorded in the RULE and DivLAB nodes, indicate of how UI elements should best be collected, shared and sequenced to facilitate the Group Agenda.

7 CONCLUSIONS AND FUTURE WORK

Our method shows potential to be a systematic and prescribed process with a solid theoretical base. We believe it will elicit meaningful Requirements from stakeholder utterances without requiring the analyst to have a deep knowledge of Activity Theory. Whilst mechanisms, heuristics and tools are still being refined, preliminary findings indicate that an AT based method can be an excellent match for complex multi user Doings. We are satisfied that AT can indeed underpin a design methodology for systems within our scope. There is indication that an end-to-end AT based method may have some advantages over some current tools and methods.

Method components for the recomposition phases and for final evaluation are beyond the scope of this paper, and will be demonstrated in future papers. A normative evaluation study of the 5-S method for a real-world system design scenario will be conducted as soon as the method components have been fully described. The evaluation will appear in future publications.

REFERENCES


LEARNING EFFECTIVE TEST DRIVEN DEVELOPMENT
Software Development Projects in an Energy Company

Wing Kum Amy Law
TransCanada, 450 – 1 Street S.W., Calgary, Alberta, T2P 4K5, Canada
amy_law@transcanada.com

Keywords: Software engineering, software development methodology, agile, test driven development, automated tests, junit tests, software design pattern, requirement analysis, software maintenance and reliable software.

Abstract: The tests needed to prove, verify, and validate a software application are determined before the software application is developed. This is the essence of test driven development, an agile practice built upon sound software engineering principles. When applied effectively, this practice can have many benefits. The question becomes how to effectively adopt test driven development. This paper describes the experiences and lessons learned by two teams who adopted test driven development methodology for software systems developed at TransCanada. The overall success of test driven methodology is contingent upon the following key factors: experienced team champion, well-defined test scope, supportive database environment, repeatable software design pattern, and complementary manual testing. All of these factors and the appropriate test regime will lead to a better chance of success in a test driven development project.

1 INTRODUCTION
TransCanada is a leader in the responsible development and reliable operation of North American energy infrastructure. TransCanada's network of approximately 41,000 kilometres (25,600 miles) of pipeline transports the majority of Western Canada's natural gas production to key Canadian and U.S. markets. A growing independent power producer, TransCanada owns, or has interests in, approximately 6,700 megawatts of power generation in Canada and the United States.

To support this enterprise, TransCanada Information System (IS) department has delivered many software solutions. Two of the solutions were Project X and Project Y. These projects were different in requirements, customers, budget, and timeline. With varying degrees, both of these teams wrote automated tests before implementation. This practice can have many benefits (McBreen, 2002):

- Identify early mistakes prior to user acceptance testing.
- Reduce time to locate mistakes in the code.
- Analogous to documentation on how to use a class.
- Increase confidence that changes in one place have not broken functionality in another place.

The question becomes how to effectively adopt test driven development. As a programmer on these industrial projects, the lessons learned will be presented to address the following how-to questions:

- How do you start a project focusing on testing first?
- How do you establish a test scope?
- How do you configure an effective test database?
- How do you reuse automated test components?
- How do you guarantee system quality?

2 TRANSCANADA IS PROJECTS

<table>
<thead>
<tr>
<th></th>
<th>Project X</th>
<th>Project Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>Test Driven Skills</td>
<td>Adopter</td>
<td>Advance</td>
</tr>
<tr>
<td>Project Duration</td>
<td>8 months</td>
<td>3 years</td>
</tr>
<tr>
<td>Database</td>
<td>Sybase</td>
<td>Oracle</td>
</tr>
<tr>
<td># of Packages</td>
<td>55</td>
<td>96</td>
</tr>
<tr>
<td># of Classes</td>
<td>1,324</td>
<td>1,560</td>
</tr>
<tr>
<td># of Unit Tests</td>
<td>1,000</td>
<td>4,350</td>
</tr>
</tbody>
</table>
As shown in Table 1, Project X and Project Y were developed by two different teams. Project X was initiated to re-engineer backend components of an existing Java web-based system to enable integration with other systems. The objective of Project Y was to replace a legacy mainframe application with a new Java web-enabled system according to prioritized business functions. These projects shared a common attribute in which both teams adopted test driven development, an agile practice built upon sound software engineering principles.

3 FOCUS ON TESTING FIRST

How do you start a project focusing on test driven development? It is difficult to introduce test driven development to programmers who are not formally trained in this area. Through the lessons learned from Project X and Project Y, a few clues will be provided on the first step towards test automation.

3.1 Project X: Early Adopters

In 2001, TransCanada IS department started to adopt agile practices in several software development projects. The programmers in Project X were the early adopters to apply test driven development in TransCanada.

Without prior experience in using test driven development practices, the programmers in Project X had to start from the beginning on every aspect. They relied on Internet articles and books to explore test driven development techniques. The programmers with fast reading speed gained advantages. They could read, absorb, and apply test driven skills through the self-learning media. The challenge was how to effectively share the knowledge.

To leverage knowledge sharing, the team adopted pair programming practice. Although pair programming is not a part of test driven development, it leads to blending expertise.

Due to personality differences, pairing was not very popular in the team. Some programmers did not passionately believe in test driven development and preferred to write code prior to writing test. They did not have automated tests for all of their code. The diversified team culture reduced the practical application of test driven development.

3.2 Project Y: Team Champion

At the other end of the spectrum, the programmers in Project Y leveraged their collective practical experience and test utility in using test driven development to facilitate their work.

They had previously been exposed to the test driven development and adhered to these practices. The team exercised pair programming to share test driven techniques. As shown in Figure 1, two programmers paired at a computer and monitor with two separate keyboards. The influence of pair programming was to cross train between programmers. With this background, automated tests were indeed written prior to implementation.

Project Y had several keen and experienced experts who built a solid foundation of test framework and set good examples for others to follow. With the supportive team culture, the automated tests typically would not be broken for longer than a day. These experts were the team champions, and they motivated everyone to adopt the practical application of test driven development.

3.3 Lessons Learned

Learning test driven development is not easy, but there are a few titbits. A champion in test driven development is a useful guide when a team is challenged. The past experience of the champion could help the reuse and extension of test utilities. Without experienced champion, programmers could become discouraged with tests that were not working for a long period of time. It is simply easier to learn new knowledge from someone who has done it before and passionately believes in it.

And yet, not every project has the luxury to find and fund an experienced team champion. When a champion is not available, reference books and Internet articles are easy-to-access learning media. Furthermore, various research and case studies are recently conducted and documented shaping the best practices. These can be conveniently circulated.
The problem with them is that they do not provide an opportunity for team collaboration.

Test driven development can be enriched through pair programming. Pair programming is like blending colours together on a paint pad, where the colours mix and influence the overall resulting colour. This is a metaphor for the blending of expertise between paired programmers; however, these benefits can be tempered somewhat where personality differences arise.

In most cases, test first guru focus on writing tests first. On the other hand, others may adapt test driven development such that both test and code are implemented in parallel. The key point is that automated tests are indeed written to ensure any changes in functionality at one place would not impact functionality at another place. This continues to provide confidence to customers who see repeatable tests pass.

4 ESTABLISH A TEST SCOPE

Assuming your project applies test driven development, how do you establish a test scope? It is impossible to test everything, and it is also suicidal to test nothing. Therefore, the fundamental principle is test things that might break (Beck, 2000). Several types of automated tests will be discussed.

4.1 Project X: Basic Principle

The programmers in Project X followed the fundamental principle, and they only wrote automated tests for things that might break. If they knew that the code was simple and it was unlikely to break, then they did not write automated tests for it. As early adopters in 2002, they had limited choices in test frameworks. The team began with only unit tests or Junit tests. A Junit test is an automated test to verify a single program or a portion of a program.

Half-way through the project, the programmers realized there was a need to validate the integration of unit components. In response, integration tests or HttpUnit tests were developed. A HttpUnit test emulates browser behaviour and allows automated tests to examine returned pages. Since the integration test concept was introduced at a later stage, the team only implemented a few integration tests. Project X was a relatively simple application, and so about 1,000 automated tests were developed.

4.2 Project Y: Test for Design

By writing tests first, the programmers in Project Y captured customer requirements and scenarios in the tests. They better understood the requirements through the realistic customer’s test data. When the tests passed, they knew that they completed the requirements. Therefore, the tests were written for design. The team began with unit tests and integration tests.

Half-way through the project, the team adopted user acceptance tests or Canoo tests. Canoo is an automated test to validate workflow. The Canoo test results are shown in colour-coded pages. The green colour represents tests passed, whereas the red colour represents tests failed. Since these Canoo tests were added at a later time, these tests were written retroactively on existing functionality and based on business priorities. Therefore, the tests were only written for a few main features. With the help of the colour-coded pages, the customers reviewed the tests in order to sign off on a release. Since Project Y had a wealth of business rules, about 4,350 automated unit tests were developed.

4.3 Lessons Learned

There are many types of software tests, such as the unit tests, integration tests, function tests, regression tests, system tests, and acceptance tests (Humphrey, 1989). It is recommended that different types of automated tests be applied to provide a wide coverage for system validation. On the other hand, it does not mean to write tests in every possible case. For example, writing tests to verify every “setter” and “getter” in a domain object is a waste of time.

Before programmers decide to implement another automated test, they should ask themselves if they gain additional business values by having it. Before they decide to stop testing, they should ask themselves four questions (Bertolino, 2001):

- What is the probability of finding more problems?
- What is the marginal cost of doing more testing to detect these problems?
- What is the probability of users encountering these problems?
- What is the impact of these problems to the users?

At a minimum, the programming team should write automated unit tests and integration tests because these tests validate the core business logic, database transactions, and interface of the overall system. Where possible, the team should also develop automated tests to validate end-to-end
system behaviour on critical features and perform manual tests to cover other areas.

The customers see the positive results from repeatable tests. This saved the customer time from extensive manual testing and became an overall cost saving. Having said that, test driven development does not deliver software more cheaply than manual approach. Therefore, establishing a test scope and seeking a balance between automation and manual approach is essential to control project cost. This is not a trivial exercise, and this topic may actually be a general interest as a paper into itself.

5 DATABASE CONFIGURATION

After you determine a test scope, how do you configure an effective test database to accommodate the test requirements and ensure test suites do not take too long to run? Database resources, administration overheads, data collision avoidance, and flexibility are key considerations in setting up a test database as examined below.

5.1 Project X: Single Database

The programmers in Project X shared a single Sybase test database. They only needed to refresh one database when there was a change in the database structure. This minimized administration overhead and database resources. However, test data could collide with one another when they executed automated tests against the same database. Hence, the tests might not pass and unwanted test data might remain in the database. The issue became exponential when multiple developers ran the automated tests at the same time as shown in Figure 2.

Although multiple test schemas in a database could accommodate the concurrency requirement, Sybase database had a technical limitation. Specifically, Sybase did not have the concept of schemas. In order to simulate multiple schemas, the team had to create multiple databases. This was not acceptable to the operational team. Thus, the team shared a single test database.

To reduce the data collisions, the programmers took an advantage of their co-location. They were seated in an open co-located area, where everyone could hear one another. They addressed this database insufficiency by announcing when a developer was about to run the automated tests such that the other programmers would not run the tests at the same time. With discipline, the team could execute all 1,000 automated tests in a single test database under thirty minutes.

Another technique was to use a private database to overcome data collision. Some developers used a private MySql open source database to setup, execute, and clean up automated test data in their own workstation. However, the team ran with the risk that the MySql open source database behaved differently than the Sybase database.

5.2 Project Y: Multiple Schemas

On the other hand, the programmers in Project Y took an advantage of Oracle database to overcome test data collision. Each programmer had a private Oracle database schema in the same database instance. Hence, anyone could create test object, execute automated test, and clean up test object at any time as shown in Figure 4. In some cases, the team used a mock object to simulate results as if a database call was made.

To expedite a unit test cycle, the team distributed automated unit test suites among four separate test database schemas. Each test cycle consisted of 4,350 automated Junit tests, and the team can execute each cycle under fifty minutes.

Two additional test database schemas were created to run HttpUnit and Canoo tests at night. This procedure was used to preserve the correct system behaviours without consuming the intensive CPU power during daytime. If a test failed, an email notification was sent out to the team.
As a result of the added convenience, the team incurred schemas administration overhead. The team accumulated up to twenty-five test database schemas. When there was a database structure change, all of schemas required a refresh.

5.3 Lessons Learned

Automated tests could generate many test databases. Some of these databases became unused due to turnover. Alternatively, a pool of test databases can be developed. When programmers are ready to execute the tests, they select available databases from the pool. After use, they can be released back to the pool.

These databases would not be identical at all times to accommodate different programming needs. The programmers could effectively execute the tests without data collision, maximize resources without increasing large administration overheads, and achieve concurrent programming. With adequate resources, automated tests should be executed every time new code is checked into the central source code repository.

6 REUSE OF TEST COMPONENT

Another interesting topic surrounding test driven development is how do you reuse automated test components? Experts do not solve every problem from first principles. Instead, they built on previous experiences making designs more flexible and ultimately reusable. A number of techniques will be addressed concerning reuse.

6.1 Project X: Design Pattern

The programmers in Project X made use of software design patterns, such as the Factory Pattern and Singleton Pattern. Since 1997, these software design patterns were well documented (Gamma, 1995). However, the maturity of software design pattern in test driven development was in the childhood stage in 2002. There were no documented reusable objects for automated tests. Perhaps it was an excuse for programmers who lacked passion to test driven development, Project X did not use any repeatable software design pattern in their automated tests. Each automated test was always a fresh new test in its own right. This saved time for the programmers by reducing learning curve to existing tests and potentially increased code readability.

6.2 Project Y: Test Design Pattern

The programmers in Project Y applied “Object Mother” software design pattern (Schum, 2001) to reuse the test data setup. The “Object Mother” pattern is a creational pattern. It aims to simplify, standardize, and maintain test object. Using this pattern, test objects can be conveniently used in any automated tests because they are created from public static methods. In case new requirements surface, any future changes can be centralized in the “Object Mother” and propagated to the children test objects. A sample of the “Object Mother” test object is shown below. In addition to “Object Mother” design pattern, the programmers made use of class inheritance such that common test methods could be shared by different automated tests.

```java
public static Pipe createPipe() {
    Pipe pipe = new Pipe();
    pipe.setId = unquieRandomNum();
    ...
    pipe.setCreatedBy = "ObjectMother";
    return pipe;
}
```

6.3 Lessons Learned

One of the challenges of test driven practices was to design, build, and effectively maintain data setup for the automated testing. The data setup involved the creation of objects required to satisfy the data constraints and test scenario. For example, a repeatable software design pattern should be used, such as the “Object Mother”, to reuse test objects. The software design pattern reduced the complexity of individual tests. It also encouraged programmers to reuse test objects in subsequent tests.

Another technique for reusing functionality in automated tests is class inheritance. This approach can group common test methods in the parent class, whereas the children classes can make use of them. Both recommendations enable programming tests less tedious to implement and change. The reusability and consistency outweigh the time invested to avoid learning curve to existing tests. There are now books, training, and internet web sites with many patterns for effective automated tests.

7 SYSTEM QUALITY

People with passion on test driven development claim that automated tests bring various benefits including system quality. So, how do you guarantee
system quality? Testing can be used to show the presence of bugs, but never their absence [4]. In spite of how much testing is performed, the team can never guarantee that an application is free of defects. The possible combinations of the input and the execution paths are too many to perform exhaustive testing. Program testing is not a simple process.

7.1 Project X: Manual Test Plan

Since the programmers in Project X did not write automated tests for the entire system, they developed a 35-page comprehensive manual test plan to validate system behaviours. The manual test plan contained test criteria and expected results to guide the users. In response to traditional software development practice, the customers performed an extensive manual user acceptance testing regimen for project sign off.

7.2 Project Y: Customer Sign Off

The programmers in Project Y had a series of automated tests that targeted to validate a wide range of business logics. These tests became a precondition for project sign off. Hence, the customers performed selective manual tests rather than extensive manual tests. As a result, the influence of test driven development practice reduced time from manual testing.

7.3 Lessons Learned

According to PMBOK (PMI, 2004), quality is the degree to which a set of inherent characteristics fulfill requirements. It is documented in requirement specifications. It can be measured by a combination of automated tests and manual tests

The automated tests should be executed as frequently as possible to reduce repetitive tests manually. They fill in the gaps incurred from manual testing. This is especially the case when the team stress level surfaced or human judgment started to degrade.

During software maintenance stage, changes to the automated tests should be made prior to changes to the source codes. Therefore, tests are always kept up-to-date with the specifications and code. This approach requires strict discipline and familiarity to the automated test architecture. Regardless of how extensive automated tests are developed, manual tests must still be performed, to some extent, in order to assure the look and feel of the system. This also increases system usability. As a result, any tuning requirements can be identified and completed prior to the production release.

Testing requires team experience and customer involvement. Therefore, the trick is to know when to stop testing, while at the same time keeping the likelihood of having the application fail post-deployment to under the target reliability objective. A balance of pair programming, code reviews, inspections, traditional manual testing, and user acceptance testing provide a complementary mechanism to test driven development for finding defects and deliver quality and reliable software.

8 CONCLUSION

There is no such thing as instant success in test driven development. However, there are clues which can enable positive results. This paper used the lessons learned from two teams to address questions surrounding test driven development. Any software development team can leverage these lessons learned and develop their own version of test driven development techniques to fit into their unique team environment. Under these conditions, we have a better chance of success in applying test driven development.

REFERENCES

TOWARDS A LANGUAGE INDEPENDENT REFACTORING FRAMEWORK

Carlos López, Raúl Marticorena
Área de Lenguajes y Sistemas Informáticos Universidad de Burgos. 09006 Burgos, Spain
{clopezno,rmartico}@ubu.es

Yania Crespo, Francisco Javier Pérez
Departamento de Informática Universidad de Valladolid. 47001 Valladolid, Spain
{yania,perez}@infor.uva.es

Keywords: Refactoring, metamodel, language independence, object oriented programming, UML.

Abstract: Using metamodels to keep source code information is one of the current trends in refactoring tools. This representation makes possible to detect refactoring opportunities, and to execute refactorings on metamodel instances. This paper describes an approach to language independent reuse in metamodel based refactoring detection and execution. We use an experimental metamodel, MOON, and analyze the problems of migrating from MOON to UML 2.0 metamodel or adapting from UML 2.0 to MOON. Some code refactorings can be detected and applied on basic UML abstractions. Nevertheless, other refactorings need information related to program instructions. “Action” concept, included in UML 2.0, is a fundamental unit of behaviour specification that allows to store program instructions and to obtain certain information related to this granularity level. Therefore, we compare the complexity of UML 2.0 metamodel with MOON metamodel as a solution for developing refactoring frameworks.

1 INTRODUCTION

Language independent refactoring is one of the current trends in refactoring research (Mens and Tourné, 2004). Defining metrics and refactorings in a language independent way offers a solution to the reuse in development of refactoring tools when they are adapted to new source languages. It is also a rational support in multilanguage integrated development environments.

There are different trends to address these problems: on the one hand, using abstract syntax trees and on the other hand, using metamodels (Demeyer et al., 1999). In this work, we display a proposal using metamodels, showing a previous support and studying the suitability of the UML 2.0 metamodel (OMG, 2004), with the new “action” concept, as new support to refactoring tools.

This paper is structured as follows: Section 2 establishes the current works with metamodels; Section 3 presents the current state of our work, describing the modules of a refactoring framework, language extensions and dependencies; Section 4 shows the standardization problem of the MOON metamodel, and presents a reengineering example, with UML 2.0 as the new candidate to replace it. Finally, in Section 5, we conclude with the pros and cons of using UML 2.0 as the core model of the refactoring framework.

2 RELATED WORKS

Some approaches to the problem of language independence are based on metamodel solutions. In (Tichelaar et al., 2000), FAMIX is defined as a metamodel for storing information, aimed at the integration of several CASE tools. One of these CASE tools is a refactoring assistant tool named MOOSE (Ducasse et al., 2000).

The FAMIX core model specifies the entities and relations that can (and should) be extracted immediately from the source code. The core model consists of the main OO entities: Class, Method, Attribute, InheritanceDefinition, etc.
For reengineering purposes, it needs two entities: Invocation and Access associations. An Invocation represents the definition of a Method calling another Method. An Access represents a method body accessing an Attribute. These abstractions are needed for reengineering tasks, such as dependency analysis, metrics computation and reengineering operations. FAMIX metamodel does not contain advanced inheritance and genericity features.

In the same way, a new solution based on metamodels is proposed in (Van Gorp et al., 2003). They propose eight additive and language-independent extensions to the UML 1.4 metamodel, which form the foundation of a new metamodel named GrammyUML. This study does not consider the Action Semantic package in UML.

Our proposal is based on MOON, Minimal Object-Oriented Notation (Crespo, 2000). MOON represents the necessary abstract constructions in refactorings definition and analysis, just like FAMIX. MOON abstractions are common to a family of programming languages: object-oriented programming languages (OOPL), statically typed with or without genericity. This is the basis for a metamodel-centered solution, with the objective of reusing it in the development and adaptation of refactoring tools.

The MOON metamodel stores: classes, relationships, variants on the type system, a set of correctness rules to govern inheritance, etc. The main difference with FAMIX relies on the type system. The core of MOON metamodel includes classes as Entity representing any concept in source code that has a Type (self reference, super reference, local variable, method formal argument, class attribute and function result). It also collects the method body description: local variables, formal arguments and instructions. The instructions are classified in the following way: creation, assignment, call and compound instructions.

For example, Figure 1 outlines the MOON metamodel classes related to genericity and their semantic rules expressed in OCL. Types are classified into formal parameters (FORMAL_PAR) and types derived from class definitions (CLASS_TYPE). Non-generic class definition leads to a 1-1 association between class (CLASS_DEF) and type (CLASS_TYPE). When class definition is generic, it is said that is the “determining class” of a potentially infinite set of types. Each generic instantiation corresponds to a different type (CLASS_TYPE). A generic class definition contains a list of formal parameters.

All these previous works propose metamodels to support a complete refactoring process. In the next section, the current state of the framework is proposed over MOON, in order to evaluate the suitability of the UML 2.0 metamodel in Section 4.

3 REFACTURING FRAMEWORK

This section establishes the current state of the proposed framework, focusing on the core (in Section 3.1) and the concrete language extensions (in Section 3.2), checked in previous works. The main purpose is to give the “big picture” of the current proposal, explaining the role of the MOON metamodel.

3.1 Framework Core

In previous works, we have designed a framework to detect, to define, and to execute refactorings. Figure 2 shows the main modules that compose it.

Here we present a brief description for each one of them:

**Module A** represents the metamodel. The metamodel is the basis of the solution for language independence. It must collect the basic elements of any object-oriented language: classes, attributes, methods, client-provider relations, inheritance and genericity. In particular, it is necessary to include information about instructions, assignment instructions and expressions.

**Module B** defines the refactoring engine. It is composed of a core and a repository. The engine core contains the necessary abstract classes to define
the refactorings by composing their inputs, pre-,
postconditions and actions. The core establishes how
to execute any concrete refactoring, once its
components are known. The refactoring repository
contains concrete predicates, functions and actions
as well as the concrete refactorings as their
composition. All the repository elements are defined
on the metamodel elements and provide the reuse
functionality when a concrete language extension is
developed.

Figure 2: Language Independent Refactoring Framework.

Module C is composed of two submodules: Metric Collector and Bad Smells Inference. It is
responsible for detecting bad smells using metrics. To remove the detected bad smells, it is possible to
suggest a refactoring set defined in the refactoring repository. In (Fowler, 2000), relations between bad
smells and refactoring have been proposed. The idea
of collecting metrics from a language independent metamodel presents clear advantages from the reuse
point of view.

Module D isolates the queries and traversals on
metamodel elements. These queries and traversals are necessary in the refactoring repository and in the
metric collector. This module is reused, avoiding
duplicated code in modules B and C. Furthermore,
Visitor and Strategy (Gamma et al., 1995) design
patterns are applied to add new operations without
changing the metamodel classes on which they
operate.

3.2 Framework Extension

The framework introduced here has been validated
with the implementation of two concrete languages
extensions: Java and Eiffel, because to reuse the
framework core, a concrete language extension is
required. In Figure 3, we show some examples of
Java extensions plugged in the framework core.

Module extension A is responsible for picking
up source code information and transforming it into
language extension instances. There are particular
language features, for example in Java, native and transient modifiers that are represented in java
extension on the metamodel core. Their classes
implement abstract methods defined on the core
module.

Module extension B defines the specific features
of each refactoring, according to the particular
selected language.

Module extension D uses language extensions to
regenerate the code. Each concrete language
extension in module extension A has the semantic to
to walk through elements using visitor classes.

4 EXAMPLE: UML 2.0 AS
METAMODEL

MOON can evolve to be fitted for a standard
metamodel. UML is currently embraced as the
standard in object oriented modelling languages.

When talking about requirements for using a

Figure 3: Refactoring Framework with Java Extensions.
metamodel in the refactoring context, a question is missing: can the method body (instruction, local variable, etc...) be stored with UML? The action concept, defined in UML 2.0, is the fundamental unit of behavior specification. An action takes a set of inputs and converts them into a set of outputs. Actions could store method body information, hence UML 2.0 metamodel can become a candidate to module A (see Figure 2). Furthermore, UML 2.0 includes template mechanisms, which provide support for generic types available in programming languages. Module A extensions could be solved with UML profiles (OMG, 2004). Profiles are mechanisms that allow metaclasses from existing metamodels to be extended to adapt them for different purposes. Profiles include the ability of tailoring the UML metamodel for different language features, such as Java, C#, C++, Eiffel, etc. The Profile mechanism is consistent with the OMG Meta Object Facility (MOF).

In the following subsections we present an example. It guides the mapping from program language constructions to UML abstractions, including advanced features: actions and templates.

### 4.1 Statement

It is very difficult to select an example that contains all language features. Our example considers the parameterized factory methods when the Abstract Factory design pattern (Gamma et al., 1995) is applied. A generic factory (GenericFactory) with subtyped bound is defined, avoiding to use an inheritance hierarchy with Factory classes to create suitable product objects.

In Figure 4, a structural solution using design patterns is shown, where the generic class with subtyped bound (GenericFactory) and generic instantiations (FactoryConcreteProd1, FactoryConcreteProd2) are highlighted. Both of them are used by client class (FactoryTest) to create new instances of concrete products (ConcreteProduct1, ConcreteProduct2).

Code 1 defines (in Java 1.5) the GenericFactory class using reflective programming by means of java.lang.Class<T> class, while Code 2 collects the piece of code associated to FactoryTest in Figure 4. In the shown codes, reserved words have been emphasized.

```java
public class GenericFactory<br>&<br>P extends ProductIF &<br>implements FactoryIF{
 private Class<P> c;
 public GenericFactory<br>(Class<P> c) {
 this.c = c;
 }
 public P createProduct(){
 P product = null;
 try{product=c.newInstance();}
 catch(InstantiationException e){ }
 catch(IllegalAccessException e){ }
 return product;
 }
}
```

Code 1: Generic Factory Definition Java 1.5

```java
FactoryIF factory1 = new GenericFactory<ConcreteProduct1>(ConcreteProduct1.class);
GenericFactory factory2 = new GenericFactory<ConcreteProduct2>(ConcreteProduct2.class);
```

Code 2: Generic Factory Instantiations.

### 4.2 UML Mapping

We outline the relevant problems of mapping a factory method createProduct body to UML abstractions: exception handlers, instruction sequences, call instructions and parametric types. In these mappings the related sections of the “UML 2.0 Superstructure” (OMG, 2004) are indicated.

To model an operation we can associate an activity diagram (Booch et al., 1999). An action flow of the operation is represented, so that all diagram elements are semantically linked to an underlying model with expressive richness.
UML 2.0 introduces new functionality, allowing to catch exceptions and manage them inside an activity diagram. In the behaviour specification, and more precisely in the activity section, we find the ExceptionHandler class. This element specifies the body (action sequence) to be executed in case that exceptions happen during the running of a protected node. A protected node groups an activity set that could throw one or more exceptions. Graphic notation of both of them can be observed in Figure 5. To map these concepts the following UML superstructure sections are used:

- ExtraStructuredActivities (in Activities) mapping Exception Handler.
- Activities mapping Instruction sequence.

In the GenericFactory class context, Figure 5 shows an activity diagram that defines the algorithm of the createProduct operation. This operation has a signature with a return value of $P$ parameter type. GenericFactory class has an attribute $c:java.lang.Class<P>$ that allows us to accomplish the creation of new concrete products.

Figure 5: Method body createProduct.

Actions can be contained in activities which provide their contexts. Activities specify control and data sequence restrictions over the actions, as well as nesting mechanisms in control structures.

Each activity is defined by a set of actions which provide precise semantics. The mapping of the instructions $c.newinstance()$ with the call action (CallOperationAction) is represented in the object diagram of Figure 6. This action obtains its input (InputPin) from the output (OutputPin) of the reading actions (ActionInputPin and ReadStructuralFeatureAction). In that way, it has the object reference contained in the attribute $c$ (StructuralFeature) in GenericFactory class.

The UML superstructure sections used to map the call instruction $c.newinstance()$ have been actions and classes.

In the UML 2.0 superstructure we can find a section focused on auxiliary constructors defining mechanism sets. One of these mechanisms is the use of templates as support to parameterize classifiers (Classifiers), packages (Packages) and operations (Operations). We can also find mechanisms to define templates, formal parameters (TemplateParameters) and to tackle the generic instance process. This subsystem has 20 classes.

Before introducing the example instantiation, we give a brief description of metaclasses and relations among the participants:

TemplateableElement is the abstraction that supports the template definition. It can contain a TemplateSignature that specifies a formal parameter sequence (TemplateParameter). TemplateableElement can also contain links to other Classifiers:TemplateableElement, related to the generic instantiations, by the substitution of the formal parameters (TemplateParameters) with the current parameters (ParameterableElement).

TemplateParameter refers to a ParameterableElement that it is exposed as a formal parameter in the template.

TemplateParameterSubstitution associates the current parameters with the formal parameters as part of the TemplateBinding relation.

RedefinableTemplateSignature specializes TemplateSignatures and RedefinableElement to allow adding new formal parameters in the definition context of templates over classifiers.

A classifier (Classifier) is a specialization of TemplateableElement and of ParameterableElement.

In Figure 7 we show the mapping of the generic factory definition with its bounded formal parameter by the subtype of ProductIF.
GenericFactory:Class:Classifier
:TemplateableElement
:TemplateSignature
:RedefinableTemplateSignature
P : TemplateParameter
ProductIF:Class : Classifier

Figure 7: Generic definition.

Figure 8 shows the generic instantiation of a concrete product factory of ConcreteProduct1. Those objects that are useful as nexus between both diagrams have been highlighted.

Figure 8: Generic instantiation.

5 CONCLUSIONS

UML can be used to store source code, even though there is a high complexity in the metamodel structure. Table 1 shows a comparison of the two metamodels, MOON and UML, divided in subsystems/sections and number of classes. The number of related classes in the UML metamodel is three times higher than in the MOON metamodel.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>MOON Number of classes</th>
<th>Sections</th>
<th>UML Number of classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Module</td>
<td>24</td>
<td>classes</td>
<td>55</td>
</tr>
<tr>
<td>Inheritance</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Genericty</td>
<td>5 templates</td>
<td>actions</td>
<td>20</td>
</tr>
<tr>
<td>Instructions</td>
<td>20 actions</td>
<td>activities</td>
<td>54</td>
</tr>
<tr>
<td>Total</td>
<td>56 Total</td>
<td></td>
<td>181</td>
</tr>
</tbody>
</table>

The paper focuses on UML abstractions which are needed to represent code information, classes and activity diagrams. Although the displayed example in Section 4 achieves a mapping to UML abstractions (generic classes, exceptions, etc.), the experiment is limited because it does not include all abstractions in object oriented languages. In this sense, we have identified some abstractions that are not represented in the UML metamodel, as typecast and multiple bounds of parametric types. Both features are supported on the MOON metamodel, but MOON does not support concepts such as exceptions, conditionals, loops, etc. Besides, MOON supports three type variants in genericity giving a suitable support to this feature in statically typed object oriented languages.

Due to the previous advantages and the minimal core size in the MOON metamodel, we think that an UML approach is only appropriate from the point of view of a standardization effort, and reuse in other abstraction levels, such as the design level. Therefore, we propose a new design solution with the UML metamodel extension as a future direction, extending the current MOON metamodel in the same way as we have done with programming languages. This should provide full support to a refactoring process reusing the previously designed framework.

REFERENCES


A DYNAMIC ANALYSIS TOOL FOR EXTRACTING UML 2 SEQUENCE DIAGRAMS

Paolo Falcarin, Marco Torchiano
Dipartimento di Automatica e Informatica (DAUIN), Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, Italy
paolo.falcarin@polito.it, marco.torchiano@polito.it

Keywords: UML, XMI, Dynamic Models, Aspect Oriented Programming (AOP), Reverse Engineering.

Abstract: There is a wide range of formats and meta-models to represent the information extracted by reverse engineering tools. Currently UML tools with reverse engineering capabilities are not truly interoperable due to differences in the interchange format and cannot extract complete and integrated models. The forthcoming UML 2.0 standard includes a complete meta-model and a well defined interchange format (XMI). There is an available implementation of the meta-model, therefore it is a viable option to use UML 2.0 the modelling format for reverse engineered models. In this paper we propose a technique to automatically extract sequence diagrams from Java programs, compliant to the UML 2.0 specifications. The proposed approach takes advantage of the Eclipse platform and different plug-ins to provide an integrated solution: it relies on a new dynamic analysis technique, based on Aspect Oriented Programming; it recovers the interactions between objects also in presence of reflective calls and polymorphism.

1 INTRODUCTION

A significant effort in the lifecycle of software systems is devoted to maintenance and comprehension. Some among the currently widespread software development practices, such as agile software development and open source projects devote less and less effort the production of documentation.

Though, documentation and design models are essential for the comprehension of software. Therefore, in order to achieve high maintainability it is important to provide developers with automatic tools to extract the documentation, consistent with the actual system.

Both in academic and industrial contexts the need for usable tools to help reverse-engineering tasks is strongly perceived. The Unified Modeling Language (OMG) is the most used standard for visual representation of the design of object-oriented software. Some UML tools provide reverse-engineering features, mainly class diagram extraction. Few tools reverse-engineer existing source code into a sequence diagram or collaboration diagram (Kollmann et al. 2001). All these tools are based on static analysis of source code instead of dynamic analysis of the executable files.

Static analysis has some limitations because it is limited to source code. In case of polymorphism, dynamic creation of objects, and using of reflection, the behaviour may vary depending on data: thus, these issues can be solved running the application, i.e. using dynamic analysis techniques. Dynamic extraction of objects interactions can be applied even with a limited knowledge of the target application.

Analyzing dynamic behaviour in such cases usually requires heavy code instrumentation and a reflective language support; moreover dealing with the huge amount of extracted information is an issue that has to be faced with; in this case it is often necessary to select the use case and the related parts of the system that should be analyzed: filters should be defined to set boundaries of the automated analysis.

Current approaches use ad-hoc debuggers or profilers to generate traces on files: these are then used to generate one or more models. These approaches differ in the kind of code instrumentation used, in the format of extracted models, and in the different tools and platforms they rely on: therefore a reverse engineering task may require time on configuration and adaptation of heterogeneous environments.
In order to extract dynamic behaviour of a software application, we present a technique based on AOP to instrument bytecode, running test cases, and then extracting sequence diagrams models, all integrated in the Eclipse platform (Eclipse). The tool we created implements several existing techniques into a single easy-to-use plug-in. In addition it is the first reverse engineering tool built on top of the UML2 Eclipse plug-in. The following sections will explain our Eclipse-based approach, the main concepts behind AOP, and will give more information on the UML2 Project (UML2).

2 THE PROPOSED APPROACH

In this section we describe the approach for extracting data from the application both statically using reflection and at run-time using AOP. The information extracted is used to populate the UML2 model of the system. The described approach has been implemented by means of some wizards working as a plug-in of the Eclipse development environment. The tool supports the following process to reverse engineer an application starting from a target Java Project in Eclipse:

1. The User identifies which java packages to inspect in the target project through a wizard in Eclipse.
2. These packages are used as parameters for a pre-defined template: the Eclipse Modeling Framework (EMF) plug-in is then used for creating an AspectTracer java file, suitable for the target application.
3. The tool changes the nature of the Java Project in an AspectJ project: therefore a new build of the target project will instrument the target application with the aspect, by means of the AspectJ weaver.
4. Throug another wizard, user selects a package with the JUnit test-cases for the target application (JUnit).
5. The wizard runs all the selected test cases and updates the UML2 model on the fly, adding a new interaction diagram for each test-case.

The result of the above mentioned analysis steps are stored in an object model using the UML2 plug-in API: this object model is based on the Eclipse Modeling Framework and therefore it can be serialized in a file in the standard XMI format, compliant with the recent UML 2.0 specification. Our approach leverage different Eclipse plug-ins to provide, in few steps, UML 2 standard models; these can be visualized and manipulated by whichever UML tool able to import XMI models, without being locked to a particular UML tool provider.

2.1 Overview of AOP

Aspect-Oriented Programming (Kiczales et al., 1997) is a new programming paradigm easing the modularization of crosscutting concerns in object-oriented software development. In particular, developers can remove scattered code related to crosscutting concerns from classes and placing them into elements called aspects. This methodology relies on a join-point model, which defines the points along the execution of a program that can be possibly addressed by an aspect. Thus, AOP involves a compiling process (called weaving) for the actual insertion of aspect code into pre-existing application source code or byte code.

AspectJ (AspectJ) is the leading AOP implementation, and the more complete, stable and widely used one; it includes a language specification, a set of additions to the Java language, a compiler that creates standard Java bytecode.

In the terminology of AspectJ an aspect is composed by a set of pointcuts and advices. The term ‘advice’ represents the implementation of a crosscutting concern, i.e. additional code to be executed in join points of the application code. AOP also involves means for identifying the join points to be extended by an aspect. The AOP term ‘pointcut’ implicitly defines at which points in the dynamic execution of the program (at which join-points) extra code should be inserted: pointcuts describe sets of join points by specifying, for example, the objects and methods to be considered, or a specific method call or execution. AspectJ offers a rich set of pointcuts: among these the ‘call’ pointcut is the more interesting for our purposes, because it intercepts method calls: the following simple example shows a simple call() pointcut, which intercepts a method call, whose signature is defined between parenthesis.

pointcut p(): call(public static void mypackage.MyClass.main(String[]));

Thus, the former pointcut, named ‘p()’, picks up a single join-point: the call to the public static method ‘main’, of class ‘MyClass’ in package ‘mypackage’, with a single parameter of type ‘String[]’ and a ‘void’ return value.

AspectJ utilizes a wildcard-based syntax to construct the pointcuts in order to capture join points...
that share common characteristics. Three wildcard notations are available in AspectJ:

1. * means any number of characters except the period.
2. .. means any number of characters including any number of periods.
3. + means any subclass or sub-interface of a given type.

Just like in Java, AspectJ provides a unary negation operator (!) and two binary operators (|| and &&) to form complex matching rules by combining simple pointcuts. The negation operator ! allows the matching of all join points except those specified by the pointcut. Combining two pointcuts with the || operator causes the selection of join points that match at least one of the pointcuts, while combining them with the && operator causes the choice of join points matching both the pointcuts.

### 2.2 Aspect Tracer

We defined the AspectTracer aspect to collect information for building the sequence diagrams in the UML2 model. In our approach a scenario is associated with a test-case, thus a use case can be related with a set of test-cases. Automated tests written with JUnit act like a sort of specification of scenarios.

Here we describe an example of AspectTracer created for the ‘Foo’ case study.

Looking at the AspectTracer’s source code (see Figure 1), the second line specifies the aspect name following the AspectJ syntax. An aspect is composed by a set of pointcuts and a set of advices.

In the aspect several pointcuts are defined and named, in order to identify different sets of join points in the application code; these pointcuts can then be composed with logical operators to define more complex pointcuts.

In order to identify these join-points, each advice is related to one named pointcut, specifying a particular set of join-points in the application code: for example, whenever the related pointcut In the aspect there are four advices of type before, used to execute some code right before the identified join-point, and one advice of type after, used to execute some code right after the join-point. Each advice contains, enclosed between braces, the additional code that is inserted at the specified join-points during the weaving process, at compile-time.

For example, test() matches a join-point in the application, the advice of type before(), at line 19, is executed immediately before the join-point.

The methodCalls() pointcut at line 9 in figure 1 can be read like this: ‘all the method calls defined in whichever package, for whichever class, whichever method, and whichever return value; moreover the wildcard “..” used between method’s parenthesis, matches whichever list of types for formal parameters.

In the same way (see line 7) we intercept all calls to constructor methods, identified by the keyword ‘new’.

Now we need to limit the scope to the ‘foo’ package, containing our case study to be inspected: we define the targetPackage() pointcut to identify all the join-points of our target application. This pointcut relies on the AspectJ pointcut within() which identifies all the join-points defined in the source code of classes matching the type pattern defined between parenthesis. For example, “within(foo..*)” matches whichever string starting with “foo” and followed by a string including periods: this identifies all the join-points defined in package “foo” and in all its sub-packages.

Wildcards are very powerful but the extensive usage made by methodCalls() pointcut, leads to pick up undesired join-points; thus, we need to define the boundary() pointcut (see line 12) to describe all join-points we want to exclude from tracing.

In particular, in order to avoid infinite recursion, we defined the instrumentation() pointcut, which excludes all the join points occurring inside our instrumentation code, i.e. the AspectTracer’s body and the related Tracer class. Moreover, we use the init() pointcut to exclude the calls to initialization methods, transparently inserted in bytecode during compilation, and occurring whenever a new object is created and its fields are initialized. Finally the callSet() pointcut (at line 15) represents the method calls we are interested to trace.

Whenever a method of a class in the “foo” package is called the related ‘before()’ advice (at line 22) is executed immediately before the join-point: this advice simply store the caller object reference. The keyword thisJoinPoint is, for an aspect, what the keyword this is for Java language, but, instead of returning the current executing object, it returns the current join-point reached along the execution. The getThis() method returns the reference of the currently executing object, advised by this aspect.
Another pointcut we used is methodExecutions() relying on the AspectJ’s execution() pointcut, which behaves like the call() pointcut: the only difference is that in this case the currently executing object (obtained with thisJoinPoint.getThis()) is the receiver object of a method call, instead of the caller object.

One may question that we just need a single before() advice for the call() pointcuts, instead of two, to extract all the information on the sender and the receiver objects. This is not possible because if we write a single advice related to a call pointcut then we are not able to extract the receiver object reference in case of constructor method calls.

On the other hand, a single advice related to an execution() pointcut would not be able to extract the sender object reference. This clarifies the need to temporarily store the sender object reference, retrieved by the call() related advices: this value will be used immediately after by the execution() related advice to invoke the Tracer tool for updating the model. At line 29, the Tracer is invoked passing these parameters: the caller object, the receiver object, the signature of the invoked method, and an array of Objects containing the parameters’ values.

It is worthwhile to notice that our approach seamlessly detects reflection-based invocations to methods and constructors (see lines 13-14). Therefore it allows identifying the actual target objects in the interaction model.

Finally it is important to notice the test() pointcut (see line 9) which intercepts the method call of a
whichever JUnit test-case, in order to load the current UML2 model before starting the test, and saving it immediately.

2.3 UML2 Sequence Diagrams

There is an ongoing effort in the Eclipse UML2 project to develop a UML2.0 compliant class library. The object model prescribed by the OMG standard is very complex, thus to make it usable the UML2 team introduced some simplifications.

A sequence diagram depicts a scenario by showing the interactions among a set of objects in temporal order. Objects are shown as vertical bars, called “lifelines”; events or message delivery is shown as horizontal arrows from the sender to the receiver (see Figure 2).

A scenario describes a typical example of an execution trace and therefore control-flow statements and conditions are not specified.

To better understand how these classes can be used to model a Java software system we present a very simple example. Let’s consider two classes A and B and we model the dynamic interaction where obj1, instance of class A, invokes method m2() of object obj2, instance of class B. This interaction can be represented by a sequence diagram as shown in Figure 2.

![Figure 2: Sample Sequence Diagram.](image)

The UML2 object-model corresponding to the sequence diagram is presented in the lower part of Figure 3. Interaction object corresponds to a sequence diagram and it contains the elements of the model: lifelines and messages. The lifelines represent instances of classes, which are represented by class Property. The messages are linked to the source and destination lifelines by two EventOccurrence objects: a send event and a receive event respectively. A message represents the invocation of a method, whose signature is represented by class Operation.

The Interaction *sd1* contains two Lifeline objects, *ll1* and *ll2*, which represent two objects obj1 and obj2, whose types are Class A and Class B respectively. The Interaction *sd1* contains a Message *msg* that is sent by Lifeline *ll1* through the EventOccurrence send and received by Lifeline *ll2* through EventOccurrence receive. The signature of the Message *msg* is the Operation *m2* of Class B.

![Figure 3: Object-model of the sample model.](image)

3 RELATED WORK

Recent research has shown that automated tools can be used to help engineers understand software systems. Commercial UML tools, and research tools extract a UML model from a system implementation. Typically, these tools use static analysis to parse the system source code or bytecode to extract a model of the system.

Shimba (Systa et al., 2001) is a reverse engineering environment to support the understanding of Java software systems. Shimba integrates the Rigi (Tilley et al., 1994) and SCED (Systa, 2001) tools to analyze and visualize the static and dynamic aspects of a subject system. The static software artifacts and their dependencies are extracted from Java bytecode and viewed as directed graph in Rigi format. The run-time information is generated by running the target software under a customized debugger, then the generated information is viewed as sequence diagrams using the SCED tool.

Jinsight (Pauw et al. 2002) is a tool for exploring a program’s run-time behaviour, by means of an ad-hoc graphical visualization based on execution traces. To collect a trace, the user runs the target program with a profiling agent and a standard JVM. Jinsight is not able to limit the trace to invocations of a particular method or class, and it has problems to scale for large code-base.
We claim that AOP usage eases reverse engineering task because code instrumentation is modularized in a single aspect that can be easily inserted or removed at build-time; moreover there is no more need of customized debuggers or ad-hoc instrumentation of source code, which are more complex to handle and error-prone.

Thanks to the aspect-oriented platform, pointcuts can be used to set precise tracing boundaries, selecting which target packages or classes to inspect and which ones to exclude.

In (Briand et al. 2005) a method to reverse engineer UML sequence diagrams from execution traces for distributed systems is described: they define how transforming extracted data in a UML 1.3 model, relying on their ad-hoc meta-model to represent sequence diagrams.

We also rely on a meta-model to generate UML models: the innovation is that we offer an integrated Eclipse environment relying on the UML2 project in order to generate models compliant with the recent UML 2 standard and exportable through XMI standard documents.

4 CONCLUSIONS

We developed and approach to model Java programs from dynamic points of view. The approach has been implemented in a working Eclipse plug-in. In summary the main highlights of the proposed approach are:

• This is the first reverse engineering approach and toolset using UML 2 as modelling infrastructure.

• It works correctly also in presence of polymorphism, allowing both a precise recovery the correct identification of invoked methods.

• Using suitable join points it is able to recognize invocations made through the Java reflection classes.

• It leverages the use of JUnit, the widespread Java unit-testing framework, to trigger scenarios executions. The test cases are formalizations of usage scenarios. This makes the proposed approach a suitable a-posteriori documentation tool for processes mainly focused on code, e.g. agile and OSS projects.

We identified several threads for further work, in particular we plan to investigate how to determine which tests are needed to obtain an acceptable coverage; then compare design sequence diagrams with the reverse-engineered ones, this will enable checking consistency between code and models made in an early design phase; finally we need to validate the overall approach with large sized software systems.

REFERENCES

MINING ANOMALIES IN OBJECT-ORIENTED IMPLEMENTATIONS THROUGH EXECUTION TRACES

Paria Parsamanesh, Amir Abdollahi Foumani
IBM Rational Software Group
Montreal, Quebec, Canada
Email: paria, amir@ca.ibm.com

Constantinos Constantinides
Department of Computer Science and Software Engineering, Concordia University
Montreal, Quebec, Canada
Email: cc@cs.concordia.ca

Keywords: Software quality, anomalies, program comprehension, aspect mining, dynamic programming.

Abstract: In the context of a computer program, the term “anomaly” is used to refer to any phenomenon that can negatively affect software quality. Examples of anomalies in object-oriented programs include low cohesion of modular units, high coupling between modular units and the phenomenon of crosscutting. In this paper we discuss the theoretical component of a technique to identifying anomalies in object-oriented implementations based on observation of patterns of messages (invoked operations). Our technique is based on the capturing of execution traces (paths) into a relational database in order to extract knowledge of anomalies in the system, focusing on potential crosscutting concerns (aspects). In order to resolve ambiguities between candidate aspects we deploy dynamic programming to identify optimal solutions.

1 INTRODUCTION

In the context of object-oriented development, the term “anomaly” refers to any phenomenon that can negatively affect the quality of software. Examples include low cohesion of modular units, and high coupling between modular units. Another example which sometimes lies at the root of the previous two quality problems is the phenomenon of crosscutting, whereby the implementation of certain concerns is not well localized but it cuts across the class hierarchy of the system, resulting in scattering of behavior and tangling of code.

In this paper, we discuss the theoretical component of a technique to identifying anomalies in object-oriented implementations. Our proposal is based on running use-case scenarios and capturing the corresponding execution traces in a relational database where we can then make observations over certain patterns of messages (invoked operations). The knowledge we extract can provide information on potential crosscutting concerns (aspect candidates). In order to identify an optimal solution while choosing an aspect among a collection of aspect candidates, we deploy dynamic programming algorithms.

The rest of this paper is organized as follows: In Section 2 we discuss the necessary theoretical background to this research. In Section 3 we discuss the problem and motivation behind this research. In Section 4 we present our proposal and in Section 5 we present our methodology. In Section 6 we discuss related work and comparisons to our proposal. We conclude our work in Section 7 with a summary, discussion and pointers to future research directions.

2 THEORETICAL BACKGROUND

The principle of separation of concerns (Parnas, 1972; Dijkstra, 1976) refers to the realization of system concepts into separate software units and it is a fundamental principle to software development. The associated benefits include better analysis and understanding of systems, readability of code, a high-level of design-level reuse, easy adaptability and good maintainability. To this end, the notions of cohesion and coupling are the fundamental measures of quality in an object-oriented software system. Cohesion is the degree to which a module performs a single responsibility (i.e. it addresses a single concern). Coupling (or dependency) is the degree to which each program module relies on another module. Two objects are said to be coupled if they act upon one another (Chidamber and Kemerer, 1991). While structuring object-oriented programs, aiming for high-cohesion and low coupling are two general goals in order to
support separation of concerns, thus providing systems which are easier to understand and maintain.

Despite the success of object-orientation in the effort to achieve separation of concerns, certain properties cannot be directly mapped in a one-to-one fashion from the problem domain to the solution space, and thus cannot be localized in single modular units. Their implementation ends up cutting across the inheritance hierarchy of the system. Crosscutting concerns (or “aspects”) include persistence, authentication, synchronization and logging. The “crosscutting phenomenon” creates two implications: 1) the scattering of concerns over a number of modular units and 2) the tangling of code in modular units. As a result, developers are faced with a number of problems including a low level of cohesion of modular units, strong coupling between modular units and difficult comprehensibility, resulting in programs that are more error prone.

Aspect-Oriented Programming (AOP) (Kiczales et al., 1997; Elrad et al., 2001) explicitly addresses those concerns which “can not be cleanly encapsulated in a generalized procedure (i.e. object, method, procedure, API)” (Kiczales et al., 1997) by introducing the notion of an aspect definition, which is a modular unit of decomposition. There is currently a growing number of approaches and technologies to support AOP. One such notable technology is AspectJ (Kiczales et al., 2001), a general-purpose aspect-oriented language, which has influenced the design dimensions of several other general-purpose aspect-oriented languages, and has provided the community with a common vocabulary based on its own linguistic constructs. In the AspectJ model, an aspect definition is a new unit of modularity providing behaviour to be inserted over functional components. This behaviour is defined in method-like blocks called advice blocks. However, unlike a method, an advice block is never explicitly called. Instead, it is activated by an associated construct called a pointcut expression. A pointcut expression is a predicate over well-defined points in the execution of the program which are referred to as join points. When the program execution reaches a join point captured by a pointcut expression, the associated advice block is executed. Even though the specification and level of granularity of the join point model differ from one language to another, common join points in current language specifications include calls to methods and execution of methods. Most aspect-oriented languages provide a level of granularity which specifies exactly when an advice block should be executed, such as executing before, after, or instead of the code defined at the associated join point. Furthermore, much like a class, an aspect definition may contain state and behavior. It is also important to note that AOP is neither limited to object-oriented programming nor to the imperative programming paradigm. However, we will restrict this discussion to the context of object-oriented programming.

3 PROBLEM AND MOTIVATION

As the complexity of software systems increases, it becomes more challenging to build software that is free of certain quality problems, collectively referred to as “anomalies.” Given an object-oriented program, we would like to be able to achieve program comprehension in order to identify anomalies at specific points over the implementation so that we can then make decisions about possible transformations such as refactoring or reengineering activities.

4 PROPOSAL

In this section we discuss our proposal to aid in the provision of high-quality systems. Several techniques have already been proposed in the literature for detecting low cohesion, high coupling and crosscutting in object-oriented implementations (Robillard and Murphy, 2001; Robillard and Murphy, 2002; Krinke and Breu, 2004; Breu and Krinke, 2004; Moldovan and Serban, 2006). In this work we build on current proposals by discussing the theoretical component of a technique to identify anomalies in object-oriented programs based on the monitoring of execution traces and the identification of patterns of invoked operations. More specifically, our proposal is based on running use-case scenarios and capturing the corresponding execution traces into a relational database. We can then apply certain strategies which can help us identify anomalies. In cases of the existence of several candidate aspects, we can deploy dynamic programming to resolve ambiguities.

5 METHODOLOGY: EXTRACTING KNOWLEDGE

In this section we discuss how our proposal is being realized.

Consider the relational database schema in Figure 1 which can store message sequences at run time. The main entities of the schema are defined as follows:

- **Class**: A class encapsulates state and behavior. Class instances (objects) send and receive messages.

- **Method**: Methods collectively make the behavior of a class. In this model we only consider those methods which appear in execution paths.
• Message: A message is an invocation of a method involving a caller (sender) object and a callee (receiver) object. A message is received and acted upon by a method inside a class definition, i.e. \(< \text{message} >::= \text{object} \cdot \text{name.operation} \cdot \text{name}\). A message implements a partial responsibility of a system operation and it forms part of a use-case scenario.

• Execution path: A use-case scenario is described as a sequence of interactions between entities. During requirements analysis, this sequence is described at a high-level of abstraction in terms of the interaction between an actor and the system where the latter is viewed as a black box. During design, a fine grained model captures sequences of interactions between class instances through an ordered sequence of message passing. We define this ordered sequence of message passing as an execution path, i.e. \(< \text{execution.path} >::=< \text{message}^+ >\).

![Relational database schema for storing execution paths.](image)

**Main idea:** Our technique to identify anomalies in object-oriented implementations is based on capturing execution paths as data from the runtime environment and populating a relational database. Data can then be analyzed in order to provide information in the form of meaningful statistics. In doing so, we can proceed as follows:

1. Generate data: Execute all possible use-case scenarios. Each use-case scenario is comprised by a sequence of execution paths.
2. Capture data: Capture execution traces as data and store them into a relational database schema.
3. Analyze data: We identify patterns of messages in execution paths. By applying certain strategies which we have developed (to be discussed next), we can identify anomalies based on the result sets.

In the next subsections we will describe the three different types of knowledge on anomalies we wish to acquire from the relational database schema, namely low cohesion, high coupling and crosscuttings. We define algorithms to identify the three types of anomalies.

5.1 Identifying Low Cohesion

We provide the following definitions:

**Definition 5.1** The term “fanin” is used to define the number of messages that a given object receives over a single execution path. The term is also defined for methods.

**Definition 5.2** The term “fanout” is used to define the number of messages that a given object sends to other objects over a single execution path. The term is also defined for methods.

**Definition 5.3** A class is defined to be independent in the context of a use-case scenario, if 1) the class forms an integral part of the use-case scenario and 2) the class is not dependent on any other classes within the use-case scenario. We can consider independent classes as potential aspects (Foumani and Constantinides, 2005a; Foumani and Constantinides, 2005b).

We would like to obtain the following knowledge:

• Classes that are involved in (short length) scenarios that have large fanin and low fanout can be considered as having low-cohesion or having an independent role in a use-case scenario.

• Methods that are involved in (short length) scenarios that have large fanin and low fanout can be considered as having low-cohesion or having an independent role in a use-case scenario. These methods can be separated from their original class through refactoring and they can be captured as advice blocks inside aspect definitions.

**Definition 5.4** We define “class independency factor” (CIF) as

\[
\text{CIF} = \frac{(\text{class fanin})}{(\text{class fanout})}
\]

In the case where \(\text{class fanout} = 0\) (in which case \(\text{CIF} \to \infty\)), the class is considered as one with an independent role and it can be a candidate aspect. To calculate this factor we need to execute statements to measure class fanin, and class fanout which is described in the subsequent subsections.

5.1.1 Measuring Class Fanin

In the following query we iterate over all messages in all execution paths where a unique identifier for a given class, \(\text{CLASS.ID}\), is provided, and we count the number of references to the class:
2. They may have an independent role (and can be po-
1. They may have many responsibilities (low cohe-
we may have the following cases:

Definition 5.5 We define “method independency fac-
we define class cohesion factor (CCH)

Definition 5.6 To identify classes with low cohesion,
CCH = (number of classes in execution path) (number of methods in execution path)
The CCH factor defines the distribution of invoked methods in an execution path between classes. In other words, CCH defines the distribution of responsibilities between classes. According to this definition we may have the following cases:

1. **CCH ≈ 1**: Logical distribution of responsibility between methods of classes. To maintain high cohesion we need to have a **CCH** value very close to 1. This would imply that the number of invoked methods and the number of classes involved in a use-case scenario should be very close to each other. We can identify an exception when there is only one class and one method referenced in the execution path. In this case, even though 

2. **CCH ≪ 1**: Many methods of a few classes are called in a given execution path. This implies that we have classes with low cohesion.

Table 1 provides different illustrative cases for calculating and interpreting **CCH**.

5.2 Identifying High Coupling

To investigate coupling and identify classes with high coupling we need to calculate two factors:
1. **CCH**: class cohesion factor. Recall that **CCH** ≈ 1 implies that many methods of a few classes are invoked in a given execution path.
2. The **fanin** of each method. This can be provided by the count of the number of invocations originated from (methods of) other classes.

Definition 5.7 For two classes \(C_i\) and \(C_j\) with methods \(M_i\) and \(M_j\) respectively, we define “class coupling” factor (**CCP**) as

\[
CCP = \frac{\sum C_i.fanin(M_i,C_j)}{\sum C_j.fanin(M_j,C_i)}
\]

or

\[
CCP = \frac{\sum fanin(C_i,C_j)}{\sum fanin(C_j,C_i)}
\]

The term \(C_i.fanin(M_i,C_j)\) refers to the number of invocations for method \(C_i\) from \(M_i\) that are originated from \(C_j\). The term \(fanin(C_i,C_j)\) refers to the number of messages received in \(C_i\) from \(C_j\). If **CCP** ≈ 1 (the summation of **fanin** of methods of class \(C_i\) is very close to the summation of **fanin** of methods of class \(C_j\)) we can say that the degree of coupling between these two classes is high. If one of these summations is zero, then we will have low coupling. If \(C_i.fanin(M_i,C_j) \gg 0\) or \(C_j.fanin(M_j,C_i) \gg 0\), then we identify another type of anomaly which we call a “high dependency” of \(C_j\) over \(C_i\) or vice versa. Table 2 provides different illustrative cases for calculating and interpreting **CCP**.

Example: Consider Figure 2 which illustrates an example execution path. Initially, we need to obtain **CCH** to see whether many methods of few classes
Table 1: Examples of class cohesion through the CCH factor with interpretations.

<table>
<thead>
<tr>
<th># of classes</th>
<th># of methods</th>
<th>CCH</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Since this is the only class in the execution path, this implies that the class has low cohesion.</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>0.2</td>
<td>Low cohesion at the class level.</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>0.1</td>
<td>Low cohesion at the class level.</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>High cohesion at the class level; Logical distribution of responsibilities between classes.</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>0.7</td>
<td>High cohesion at the class level. In this case we still may have low cohesion at the method level; To identify low cohesion at the method level we need to calculate the MIF factor for each method. We consider methods with high MIF value as candidate methods with low cohesion.</td>
</tr>
</tbody>
</table>

are invoked during execution. CCH is obtained as follows:

\[
CCH = \frac{2}{8} = 0.25 \ll 1
\]

The very small value of CCH indicates a potential high coupling between the involved classes. To investigate high coupling further, we must read the CCP value, which can be obtained as follows:

\[
\sum C_i.fanin(M_i,C_j) = fanin(C_i,M_1,C_j) + fanin(C_i,M_2,C_j) + fanin(C_i,M_3,C_j) + fanin(C_i,M_4,C_j) = 0 + 1 + 1 + 2 = 4
\]

\[
\sum C_j.fanin(M_j,C_i) = fanin(C_j,M_5) + fanin(C_j,M_6) + fanin(C_j,M_7) = 2 + 1 + 1 = 4
\]

\[
CCP = \frac{4}{4} = 1
\]

The value of CCP indicates that there is indeed high coupling between the two classes C_i and C_j.

Figure 2: Illustration of high coupling.

5.3 Identifying Crosscutting

To identify crosscutting we first need to obtain a set of execution paths. Consider the following:

<path1> ::= ...
<path2> ::= ...
<path3> ::= ...
<path4> ::= ...
<path5> ::= ...
<path6> ::= ...

MINING ANOMALIES IN OBJECT-ORIENTED IMPLEMENTATIONS THROUGH EXECUTION TRACES

181
Table 2: Interpretation of class dependency.

<table>
<thead>
<tr>
<th>$C_i.fanin(M_i, C_j)$</th>
<th>$C_j.fanin(M_j, C_i)$</th>
<th>CCP</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\gg 1$</td>
<td>$\infty$</td>
<td>High dependency of $C_i$ over $C_j$.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\infty$</td>
<td>Low coupling.</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>$\approx 1$ or $&gt; 1$</td>
<td>High coupling.</td>
</tr>
</tbody>
</table>

\[ \sum C_i.fanin(M_i, C_j) \approx \sum C_j.fanin(M_j, C_i) \]

\[ \text{CCP} \]

\[ \text{Interpretation} \]

\[ \text{0} \gg \text{1} \]

\[ \text{High dependency of } C_i \text{ over } C_j. \]

\[ \text{0} \]

\[ \text{Low coupling.} \]

\[ \text{6} \approx \text{7} \text{ or } > 1 \]

\[ \text{High coupling.} \]

A crosscutting path, indicated by $S_k$, is one whose sequence of messages occurs in several execution paths. Consider the following crosscutting paths based on the above execution paths $S_1$–$S_{10}$:

$\text{Sa(path 1,2,3)}$:

\[ \langle \text{obj1.op1, obj2.op2, ...} \rangle \]

$\text{Sb(path 4,5,6)}$:

\[ \langle \text{obj4.op4, obj5.op5, obj6.op6, ...} \rangle \]

$\text{Sc(path 7,8)}$:

\[ \langle \text{obj4.op4, obj5.op5, obj6.op6, ...} \rangle \]

$\text{Sd(path 9,10)}$:

\[ \langle \text{obj3.op3, obj4.op4, obj5.op5, obj6.op6, ...} \rangle \]

A crosscutting path, indicated by $S_k$, is one whose sequence of messages occurs in several execution paths. Consider the following crosscutting paths based on the above execution paths $S_1$–$S_{10}$:

$\text{Sa(path 1,2,3)}$:

\[ \langle \text{obj1.op1, obj2.op2, ..., obj4.op4, obj5.op5} \rangle \]

$\text{Sb(path 4,5,6)}$:

\[ \langle \text{obj4.op4, obj5.op5, obj6.op6} \rangle \]

$\text{Sc(path 7,8)}$:

\[ \langle \text{obj4.op4, obj5.op5, obj6.op6} \rangle \]

$\text{Sd(path 9,10)}$:

\[ \langle \text{obj3.op3, obj4.op4, obj5.op5, obj6.op6} \rangle \]

We define $S$ as the set of all sequences $S_k$. Essentially $S$ is a set of candidate aspects. Aspect candidates should not contain common operations because this will violate the semantics (logic) of the initial object-oriented system. As a result, intersections between the aspect candidate sets constitute redundancies. We, therefore, need to identify which one among those sets (or subsets of them) can constitute better candidate aspect sets. In the following subsections we describe an algorithm in order to identify aspect candidates in situations when selecting aspect candidates is ambiguous.

### 5.3.1 Identifying Orthogonal Aspect Candidate Sets

This subsection presents an algorithm to address the issue of redundancies and the identification of strong candidate aspects. The goal is to eliminate redundancies by identifying aspect candidate sets that do not have common operations. The algorithm has two steps. In the first step the algorithm eliminates the longest common sets from the list of message sequences. The longest common set (there could be more than one) is one which can be obtained by the union of others. As a result, we can remove the longest common sets from $S$, the set of aspect candidates. In the second step the algorithm finds orthogonal sets. An orthogonal set is one which does not have any common element with any other set.

Suppose $S = \{S_1, S_2, S_3, ..., S_x, ..., S_n\}$ is a set of message sequences and we suppose $S_d$ is a message sequence in the set with the longest common set. The first condition satisfies that a given set has common elements with some of the other message sequences. The second condition satisfies that $S_d$ can be covered by the union of other message sequences. We can now define set $S'$ as a new set of aspect candidates by removing $S_d$ from set $S$. We repeat this process for all the other members of the aspect candidate set until there is no member that can be covered by a union of other members. In the example, we identify $S_d$ as the longest common set (Figure 3), since:

$$
\exists S_i \in S' : S_d \bigcap S_i = \emptyset
$$

The first condition satisfies that a given set has common elements with some of the other message sequences. The second condition satisfies that $S_d$ can be covered by the union of some of the other message sequences. We can now define set $S'$ as a new set of aspect candidates by removing $S_d$ from set $S$. We repeat this process for all the other members of the aspect candidate set until there is no member that can be covered by a union of other members. In the example, we identify $S_d$ as the longest common set (Figure 3), since:

$$
S_d \bigcap S_n = \emptyset
$$

$$
S_d \bigcap S_i = \emptyset
$$

$$
S_d = S_a \bigcup S_c
$$

Figure 4 illustrates $S$, the set of all message sequences and set of candidate aspects, after having removed $S_d$, the longest common message sequence.
2. Define $C'_i$ as a set of all members of $S_x$ that are not in $S_i$.

$$S_i - C_i = C'_i, \forall S_i \in S'$$

3. Define $S'_x$ as a set of $S_x$ members except the members that belong to $C_i$.

$$S'_x = S_x - \bigcup_{i=1}^{m} C_i, \forall S_i \in S'$$

4. Replace set $S$ with a new set of aspect candidates such that each $S_i$ in $S$ is replaced by $C_i$ and $C'_i$ and also $S_a$ replace with $S'_a$.

5. We repeat steps 1 to 4 until we have reached the following condition:

$$S_i \cap S_j = \emptyset, \forall S_i, S_j \in S \text{ and } i \neq j$$

This condition satisfies that there are no two sets with common members in $S$. In this case, the members of $S$ can be defined as aspect candidates.

Applying this algorithm to the example (Figure 4) where $S = \{ S_a, S_b, S_c \}$, and starting with sequence $S_a$, we obtain the following:

1. Find common members of $S_a$ with $S_b$ and $S_c$ as $C_b$ and $C_c$ sets:

$$C_b = S_a \cap S_b = \{ op2 \}$$

$$C_c = S_a \cap S_c = \{ op4, op5 \}$$

2. Define $C'_b$ and $C'_c$ as sets of all members of $S_b$ and $S_c$ that are not in $S_a$.

$$C'_b = S_b - C_b = \{ op1 \}$$

$$C'_c = S_c - C_c = \{ op6 \}$$

3. Define $S'_a$ as a set of $S_a$ members except the members that belong to $C_b$ and $C_c$.

$$S'_a = S_a - (C_b \cap C_c) = \{ op3 \}$$

4. Replace set $S$ with a new set of aspect candidates such that each $S_b$ in $S$ is replaced by $C_b$ and $C'_b$, $S_c$ in $S$ is replaced by $C_c$ and $C'_c$, and $S_a$ replace with $S'_a$.

$$S = \{ S'_a, C_b, C'_b, C_c, C'_c \}$$

5. Since $S$ holds orthogonal sets, we do not need to repeat the algorithm.

In Figure 4 we identify $A_1$–$S$ as orthogonal sets, all these being potential aspects. In general, it is highly unlikely that aspects are completely orthogonal of each other. When aspects affect the same join point in one particular sequence, then the order of aspect (advice) execution is important, as this must preserve the semantics of the original object-oriented
system. For example for the aspect candidates in Figure 4, aspects should be defined with the following precedence:

\[ A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4 \rightarrow A_5 \]

This algorithm identifies a set of aspects by breaking down the clone message sequences into the lowest level of their method invocations such that there are no conflicts between these sets. However, we believe that this cannot be an optimal solution for the problem since it does not consider the following parameters:

1. Number of identified aspects.
2. Number of different types of concerns addressed by each aspect definition. An aspect definition may be defined with low cohesion.
3. Number of messages in a given execution path.
4. Number of execution paths in a given use-case scenario.
5. Number of runs of all execution paths.

In order to provide optimal solution for the second step of the algorithm, we deploy dynamic programming which is discussed in the next subsections.

5.3.2 Redundancy of Adjacent Invocations (Rai)

In this subsection we introduce a dynamic programming algorithm to identify aspect/advice candidates based on a set of given parameters. Dynamic programming (Cormen et al., 2002) is typically applied to optimization problems. In such problems there can be many possible solutions. Each solution has a value, and we wish to find a solution with the optimal (minimum or maximum) value. In this example, we wish to define parts of message sequences as an aspect (such that the selected operations can be modeled as an advice block) in such a way that the message sequence satisfies the following conditions:

1. It is used in the maximum number of paths.
2. It has the maximum RAI value.
3. It contains the minimum number of classes.
4. It contains the minimum number of operations.

We believe this strategy can identify a stronger set of aspect candidates since it selects the most utilized operations. By minimizing the number of classes and operations we achieve a better modularity. We initially define two new factors to detect clone message sequence and also we introduce new algorithms to identify aspect candidates.

**Definition 5.8** We define "redundancy of adjacent invocations" (RAI) of a given execution path to be the ratio of the recurrence of a sequence \( S_t \) of adjacent messages, over the total number of execution paths.

\[
\text{RAI}(\text{path}) = \frac{\text{recurrence of a message sequence}}{\text{total number of paths}}
\]

As a result, for each execution path in the example of Section 5.3 we have the following RAI measures:

\[
\text{RAI}(S_a) = \frac{3}{10}, \quad \text{RAI}(S_b) = \frac{3}{10}, \quad \text{RAI}(S_c) = \frac{2}{10}
\]

From the set \( S = \{S_a, S_b, S_c\} \), we need to select a message sequence or sequences that will be the best candidate for an aspect. Table 3 provides illustrative cases for the following parameters:

- NC: Number of classes in a given execution path.
- NO: Number of different operations in a given execution path.
- NX: Number of execution times of given execution paths.

<table>
<thead>
<tr>
<th>Message Sequence</th>
<th>NC</th>
<th>NO</th>
<th>NX</th>
<th>RAI</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_a )</td>
<td>4</td>
<td>4</td>
<td>1000</td>
<td>( \frac{3}{10} )</td>
</tr>
<tr>
<td>( S_b )</td>
<td>2</td>
<td>2</td>
<td>100</td>
<td>( \frac{3}{10} )</td>
</tr>
<tr>
<td>( S_c )</td>
<td>2</td>
<td>3</td>
<td>100</td>
<td>( \frac{2}{10} )</td>
</tr>
</tbody>
</table>

According to this recurrence equation, we select \( S_x \) that does not have any common member with the other \( S_i \)'s in set \( S \). For \( S_i \)'s that have common
members, we recursively select \( S_x \) that has a maximum \( \text{NX} \) value, maximum \( \text{RAI} \) value, minimum \( \text{NC} \) value and minimum \( \text{OP} \) value. For the selected \( S_x \) we apply the following formula to identify aspect candidates:

\[
S_x = S_x - \bigcup_{i=1}^{m} S_i, \forall S_i \in S'
\]

In the example, by applying this algorithm we select \( S_a \) as the first candidate aspect (as it has \( \text{MAX} \langle \text{NX} \rangle \)). The second aspect candidate according to Table 3 would be \( S_b \) and since there is no intersection between \( S_a \) and \( S_b \), we identify the latter as a new aspect candidate. The last aspect candidate, \( S_c \), has an intersection with both \( S_a \) and \( S_b \). For this reason we apply the above formula for \( S_c \) to obtain a strong aspect candidate, i.e.

\[
S_c = S_c - (S_a \cup S_b) = \text{op3}
\]

Our RAI optimization strategy has decreased the number of candidate aspects from five down to three.

### 5.3.3 Family Related Dependency (Frd)

Consider the following paths that contain method invocations in the same class, disregarding any possible gaps between them:

- \( \langle \text{path1} \rangle := \langle \ldots, \text{obj1.op1}, \ldots, \text{obj1.op2}, \ldots, \text{obj1.op3}, \ldots \rangle \)
- \( \langle \text{path2} \rangle := \langle \ldots, \text{obj1.op1}, \ldots, \text{obj1.op2}, \ldots \rangle \)
- \( \langle \text{path3} \rangle := \langle \ldots, \text{obj2.op1}, \ldots, \text{obj2.op2}, \ldots, \text{obj1.op3}, \ldots \rangle \)
- \( \langle \text{path4} \rangle := \langle \ldots, \text{obj1.op1}, \ldots, \text{obj1.op2}, \ldots, \text{obj2.op3}, \ldots \rangle \)
- \( \langle \text{path5} \rangle := \langle \ldots, \text{obj1.op1}, \ldots, \text{obj1.op2}, \ldots, \text{obj2.op3}, \ldots \rangle \)
- \( \langle \text{path6} \rangle := \langle \ldots, \text{obj1.op5}, \ldots, \text{obj1.op6}, \ldots, \text{obj1.op7}, \ldots, \text{obj1.op8}, \ldots \rangle \)
- \( \langle \text{path7} \rangle := \langle \ldots, \text{obj1.op5}, \ldots, \text{obj1.op6}, \ldots, \text{obj1.op7}, \ldots, \text{obj1.op8}, \ldots \rangle \)
- \( \langle \text{path8} \rangle := \langle \ldots, \text{obj1.op9}, \ldots, \text{obj1.op10}, \ldots, \text{obj1.op11}, \ldots \rangle \)
- \( \langle \text{path9} \rangle := \langle \ldots, \text{obj1.op9}, \ldots, \text{obj1.op10}, \ldots, \text{obj1.op11}, \ldots \rangle \)

We define \( S_k \) as a sequence of messages that occurs in several paths. For example,

- \( \text{Sa}(\text{path} 1, 3): \langle \text{obj1.op1}, \text{obj1.op2}, \text{obj1.op3} \rangle \)
- \( \text{Sa}(\text{path} 1, 1, 3): \langle \text{obj1.op1}, \text{obj1.op2} \rangle \)
- \( \text{Sa}(\text{path} 4, 5): \langle \text{obj1.op1}, \text{obj1.op2}, \text{obj1.op3} \rangle \)
- \( \text{Sa}(\text{path} 6, 7, 8): \langle \text{obj1.op5}, \text{obj1.op6}, \text{obj1.op7}, \text{obj1.op8} \rangle \)
- \( \text{Sa}(\text{path} 9, 10): \langle \text{obj1.op4}, \text{obj1.op5}, \text{obj1.op11} \rangle \)

**Definition 5.9** We define “family-related dependency” (Frd) on a given execution path as the ratio of the recurrence of a sequence \( S_k \) of messages sent to the same class, over the total number of execution paths.

\[
\text{FRD}(\text{path}) = \frac{\text{number of selected paths}}{\text{total number of paths}}
\]

As a result, for each execution path we have the following FRD measures:

\[
\begin{align*}
\text{FRD}(S_a) &= \frac{2}{10} \\
\text{FRD}(S_b) &= \frac{3}{10} \\
\text{FRD}(S_c) &= \frac{2}{10} \\
\text{FRD}(S_d) &= \frac{3}{10} \\
\text{FRD}(S_e) &= \frac{2}{10}
\end{align*}
\]

From the set \( S = \{ S_a, S_b, S_c, S_d, S_e \} \), we need to select a sequence or sequences that will be the best aspect candidate. Identifying orthogonal aspect algorithm as mentioned before cannot find the optimal set of aspects. In the case of family independent dependency, we introduce another dynamic programming algorithm that uses the following parameters. Table 4 provides illustrative cases for these parameters:

1. Number of different classes in each message sequence (\( \text{NC} \))(this parameter in \( \text{FRD} \) is always equal to 1).
2. Number of different operations in each message sequence (\( \text{NO} \)).
3. \( \text{FRD} \) value for message sequences.

We define the following recurrence equation to select the next aspect candidate from Set \( S \) and we define \( S' \) as a new set of aspect candidates. Each time we will add \( S_x \) that is selected from the following formula into Set \( S' \):

\[
S_x = \left\{ \begin{array}{ll}
S_x & | S_x, \text{obj} \notin \bigcup S_j, \forall S_j \in S, j \neq i \\
\text{Max}(\text{FRD}), & \text{Min}(\text{NO}) \text{ otherwise}
\end{array} \right.
\]

This algorithm selects all the message sequences that use an object that is not used in any other message sequences. For example, \( S_c \) in Table 4 uses \( \text{obj2} \) that is not used in the other message sequences. For the other message sequences we recursively select the
We apply the above formula for \( S \) aspect candidate:

\[
S_x = S_x - \bigcup_{i=1}^{m} S_i, \quad \forall S_i \in S'
\]

In the example, by applying this algorithm we select \( S_c \) as the first candidate aspect (as \( S_c.obj(=obj2) \notin \bigcup(S.obj - S_c.obj)(= obj1) \)). The second aspect candidate according to Table 4 would be \( S_b \). We apply the above formula for \( S_b \) to obtain a strong aspect candidate:

\[
S_b = S_b - S_c = \{obj1.op1, obj1.op2\}
\]

As the third aspect candidate we identify \( S_d \) that has \( \text{MAX}(\text{FRD}) \):

\[
S_d = S_d - (S_c \cup S_b)
\]

\[
= \{obj1.op5, obj1.op6, obj1.op7, obj1.op8\}
\]

Next aspect candidate is \( S_a \) or \( S_e \) since both has the same \( \text{FRD} \) and \( \text{NO} \). In this case, we will select one of the message sequences randomly. For example, we select \( S_e \):

\[
S_e = S_e - (S_c \cup S_b \cup S_d)
\]

\[
= \{obj1.op9, obj1.op10, obj1.op111\}
\]

and finally we apply the above formula for \( S_a \), since \( S_a \) has some common members with \( S_b \), we just select the members in \( S_a \) that are not in \( S_b \).

\[
S_a = S_a - (S_c \cap S_b \cap S_d \cap S_e)
\]

\[
= \{obj1.op3\}
\]

### 5.3.4 Combining Strategies: Rai and Frd

In Section 5.3.2 we described that with the calculation of the RAI metric we are able to identify aspects by detecting clones based on method invocations. These methods belong to the different classes and are called one after another in order to play a specific part of the algorithms in the different scenarios. We claim that we can extract these methods, and model them as an aspect. In Section 5.3.3, we discussed how to detect clones based on method invocations such that the methods belong to the same class but can be called in different scenarios in the same order and pattern. Family-related dependency methods can be modeled as different advices of an aspect. In this subsection we combine RAI and FRD. Consider the execution paths illustrated in Table 5 (the result of calculating RAI values) and in Table 6 (the result of calculating FRD values). In Table 7 we show \( \text{FRD} \) and \( \text{RAI} \) as well as certain other parameters such as the number of classes, and the number of operations.

\[
<\text{path1}> ::=\quad <\ldots, \text{obj1}.op1, \text{obj2}.op2, \ldots, \text{obj1}.op11, \ldots, \text{obj2}.op22, \text{obj1}.op111, \text{obj3}.op3, \ldots>
\]

\[
<\text{path2}> ::=\quad <\ldots, \text{obj1}.op1, \text{obj2}.op2, \ldots, \text{obj1}.op11, \ldots, \text{obj2}.op22, \text{obj1}.op111, \text{obj3}.op3, \ldots>
\]

\[
<\text{path3}> ::=\quad <\ldots, \text{obj1}.op1, \text{obj2}.op2, \ldots, \text{obj1}.op11, \ldots, \text{obj2}.op22, \text{obj1}.op111, \text{obj3}.op3, \ldots>
\]

We introduce a three-step algorithm to identify aspects in these cases:

**Step 1:** We deploy the \( \text{FRD} \) algorithm to identify aspects and their corresponding advices. In our example, we identify the following aspect/advises:

- Aspect(Advice[\text{obj3}.op3])
- Aspect(Advice[\text{obj2}.op22])
- Aspect(Advice[\text{obj1}.op11])
- Aspect(Advice[\text{obj1}.op11])

**Step 2:** We remove operations selected in the first step from sets identified for \( \text{RAI} \). In this case we will have three different situations:

1. Selected operations at the beginning of a message sequence in the \( \text{RAI} \) set. In this case we can remove them from the set. The remaining operations in the set will define a new set.

<table>
<thead>
<tr>
<th>Message Sequence</th>
<th>NC</th>
<th>NO</th>
<th>FRD</th>
<th>obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_a )</td>
<td>1</td>
<td>3</td>
<td>( \frac{2}{15} )</td>
<td>obj1</td>
</tr>
<tr>
<td>( S_b )</td>
<td>1</td>
<td>2</td>
<td>( \frac{1}{15} )</td>
<td>obj1</td>
</tr>
<tr>
<td>( S_c )</td>
<td>1</td>
<td>2</td>
<td>( \frac{2}{15} )</td>
<td>obj2</td>
</tr>
<tr>
<td>( S_d )</td>
<td>1</td>
<td>4</td>
<td>( \frac{1}{15} )</td>
<td>obj1</td>
</tr>
<tr>
<td>( S_e )</td>
<td>1</td>
<td>3</td>
<td>( \frac{2}{15} )</td>
<td>obj1</td>
</tr>
</tbody>
</table>

### Table 4: Parameters for \( \text{FRD} \).
Table 5: RAI metric for paths.

<table>
<thead>
<tr>
<th>Path</th>
<th>Recurring Sequence</th>
<th>RAI</th>
</tr>
</thead>
<tbody>
<tr>
<td>{path1, path2, path3}</td>
<td>S1: {obj1.op1, obj2.op2}</td>
<td>$\frac{3}{3}$</td>
</tr>
<tr>
<td>{path1, path2, path3}</td>
<td>S2: {obj2.op22, obj1.op111}</td>
<td>$\frac{3}{3}$</td>
</tr>
<tr>
<td>{path1, path3}</td>
<td>S3: {obj2.op22, obj1.op111, obj3.op3}</td>
<td>$\frac{2}{3}$</td>
</tr>
</tbody>
</table>

Table 6: FRD metric for paths.

<table>
<thead>
<tr>
<th>Path</th>
<th>Recurring Sequence</th>
<th>FRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>{path1, path2, path3}</td>
<td>S4: {obj1.op1, obj1.op11}</td>
<td>$\frac{3}{3}$</td>
</tr>
<tr>
<td>{path1, path3}</td>
<td>S5: {obj1.op1, obj1.op11, obj1.op1}</td>
<td>$\frac{2}{3}$</td>
</tr>
<tr>
<td>{path1, path3}</td>
<td>S6: {obj3.op3}</td>
<td>$\frac{2}{3}$</td>
</tr>
<tr>
<td>{path1, path2, path3}</td>
<td>S7: {obj2.op2, obj2.op22}</td>
<td>$\frac{3}{3}$</td>
</tr>
</tbody>
</table>

Table 7: RAI & FRD metrics for paths.

<table>
<thead>
<tr>
<th>Path</th>
<th>RAI</th>
<th>FRD</th>
<th>NC</th>
<th>NO</th>
<th>Obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1: {obj1.op1, obj2.op2}</td>
<td>$\frac{3}{3}$</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>S2: {obj2.op22, obj1.op11}</td>
<td>$\frac{3}{3}$</td>
<td>-</td>
<td>2</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>S3: {obj2.op22, obj1.op111, obj3.op3}</td>
<td>$\frac{2}{3}$</td>
<td>-</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>S4: {obj1.op1, obj1.op11}</td>
<td>-</td>
<td>$\frac{3}{3}$</td>
<td>1</td>
<td>2</td>
<td>obj1</td>
</tr>
<tr>
<td>S5: {obj1.op1, obj1.op11, obj1.op1}</td>
<td>-</td>
<td>$\frac{2}{3}$</td>
<td>1</td>
<td>3</td>
<td>obj1</td>
</tr>
<tr>
<td>S6: {obj3.op3}</td>
<td>-</td>
<td>$\frac{2}{3}$</td>
<td>1</td>
<td>1</td>
<td>obj1</td>
</tr>
<tr>
<td>S7: {obj2.op2, obj2.op22}</td>
<td>-</td>
<td>$\frac{3}{3}$</td>
<td>1</td>
<td>2</td>
<td>obj1</td>
</tr>
</tbody>
</table>
2. Selected operations at the end of a message sequence in the \textit{RAI} set. In this case we can remove them from the set the same as case 1.

3. Selected operations are in the middle of a message sequence in the \textit{RAI} set. In this case we can divide the \textit{RAI} set into two sets (or more) by removing selected operations from step 1.

\textbf{Step 3}: We deploy the \textit{RAI} algorithm to the new set of message sequences. Here, we stress the importance of maintaining the precedence of aspect execution in order to preserve the semantics of the original object-oriented system.

\section{6 RELATED WORK}

A number of studies of software metrics such as coupling and cohesion to evaluate the complexity of software have been carried out. However, none of these proposals directly address the calculation software metrics at run-time in order to identify aspects.

In (Gupta and Rao, 2001), the authors measure module cohesion in legacy software. They compared statically calculated metrics against a program execution based approach of measuring the levels of module cohesion. In (Mitchell and Power, 2003), the authors adapt two common object-oriented metrics, coupling and cohesion, and apply them at run-time. In (Moldovan and Serban, 2006), the authors describe a new approach in aspect mining that uses clustering to identify the methods that have the code scattering symptom. In this method, for a method, they consider a large numbers of calling methods and a large numbers of calling classes as indications of code scattering. In order to group the best methods (candidates) they use a vector-space model for defining the similarity between methods. In (Brez and Krinke, 2004), the authors describe an automatic dynamic aspect mining approach which deploys program traces generated in different program executions. These traces are then investigated for recurring execution patterns based on different constraints, such as the requirement that the patterns have to exist in a different calling context in the program trace. In (Krinke and Breu, 2004), the authors describe an automatic static aspect mining approach, where the control flow graphs of a program are investigated for recurring executions based on different constraints, such as the requirement that the patterns have to exist in a different calling context. In (Robillard and Murphy, 2002), the authors introduce a concern graph representation that abstracts the implementation details of a concern and it makes explicit the relationships between different elements of the concern for the purpose of documenting and analyzing concerns. To investigate the practical tradeoffs related to this approach, they developed the Feature Exploration and Analysis tool (FEAT) that allows a developer to manipulate a concern representation extracted from a Java system, and to analyze the relationships of that concern to the code base. In (Robillard and Murphy, 2001), the authors describe concerns based on class members. This description involves three levels of concern elements: use of classes, use of class members, and class member behavior elements (use of fields and classes within method bodies). Use of classes is expressed by the class-use production rules. The rules specify that a concern either uses the entire class to implement its behavior or only part of a class, as well as what parts of the class participate in the implementation of the particular concern. In (Bruntink, 2004), the authors define certain clone class metrics that measure known maintainability problems such as code duplication and code scattering. Subsequently, these clone class metrics are combined into a grading scheme designed to identify interesting clone classes for the purpose of improving maintainability using aspects. In (Baxter et al., 1998), the authors use an abstract syntax tree (AST) to detect duplicated code (clones). This technique uses parsers to first obtain a syntactical representation of the source code, typically an AST. The clone detection algorithms then search for similar subtrees in the AST. In (Jr., 2002), the authors introduce a general-purpose, multidimensional, concern-space modeling schema that can be used to model early-stage concerns.

\section{7 CONCLUSION AND FUTURE WORK}

In this paper we described the theoretical component of an approach to identifying anomalies in object-oriented implementations based on observations of patterns of messages in a legacy object-oriented system. The term “anomaly” is used to refer to any phenomenon that can negatively affect software quality such as low cohesion, high coupling and crosscutting. Our technique is based on the capture of execution traces (paths) into a relational database in order to extract knowledge of anomalies in the system. We developed strategies to identify anomalies. In the case of ambiguities in the presence of multiple candidate aspects, we deployed dynamic programming to identify optimal solutions in order to group the strongest aspect candidates. We believe that our work can aid developers to find potential anomalies in object-oriented systems. Equally, the work can aid maintainers.

For future work, we intend to provide an implementation through a case study. A tracing mechanism (perhaps deploying AOP) can be used to capture execution traces into a relational database schema.
SQL queries executed over the relational database schema should be used to calculating the parameters of the various algorithms. We also plan to investigate the use of multidimensional schema, data warehouse, and data mining techniques to discover knowledge by considering different dimensions.

REFERENCES


A DETECTION METHOD OF FEATURE INTERACTIONS FOR TELECOMMUNICATION SERVICES USING NEW EXECUTION MODEL

Sachiko Kawada, Masayuki Shimokura and Tadashi Ohta
Information Systems Science, Soka University 1-236, tangi-cho, hachioji-city, Tokyo, 192-8577, Japan
ohta@t.soka.ac.jp

Keywords: Feature interaction, detecting interactions, seeming interaction, specification execution model.

Abstract: A service, which behaves normally, behaves differently when initiated with another service. This undesirable behavior is called a feature interaction. In investigating the international benchmark for detecting interactions in telecommunication services, it was found that many interactions that do not actually occur (called: "seeming interactions" in this paper) were mis-detected. The reason for mis-detection of seeming interactions is that interactions were detected using a state transition model which does not properly represent the process flow in a real system. Since seeming interactions cause an increase in time taken for solving interactions, avoiding mis-detection is an important issue. In this paper, a problem in implementing a detection system without mis-detecting seeming interactions is clarified and its solution is proposed. In addition, a new interaction detection method, which adopts the proposed solution and is based on a specification execution model which properly reflects the process flow in a real system, is proposed.

1 INTRODUCTION

A service, which behaves normally, behaves differently when initiated with another service. This undesirable behavior is called a feature interaction (hereafter abbreviated as an interaction) (Cameron, 1994). Many approaches, that formally and automatically detect feature interactions among given telecommunication services specifications, have been proposed (Amyot, 2003).

However, in investigating interactions that were described in the international benchmark for detecting interactions in telecommunication services (Griffeth, 2000), it was found that many interactions that do not actually occur (which are called "seeming interactions" in this paper) were mis-detected.

The authors have proposed a Trigger Point Model, (abbreviated as a “TP model”) as a new specification execution model which properly reflects the process flow in a real system. They have also confirmed its effectiveness (Shimokura, 2004). In implementing a detection system based on the TP model without mis-detecting seeming interactions, a change of the meaning of an event causes a problem. To solve this problem, this paper proposes a method for identifying the meaning of an event and a new interaction detection algorithm based on the proposed method and the TP model, and confirmed that the proposed algorithm is effective.

In section 2, a concrete example of a seeming interaction caused by a change of the meaning of an event is explained. In section 3, the TP model that is a basis of this paper is briefly described. In section 4, a problem in implementing a detection system is described, and a method for identifying the meaning of an event is proposed as a solution. In section 5, a new detection algorithm for interactions based on the proposed model and the TP model is proposed. In section 6, the proposed algorithm is evaluated.

2 SEEMING INTERACTION

It is well known that telecommunication services specifications can be described as state transition diagrams. In this paper, hereafter, ‘specification’ means individual state transitions in the state transition diagram for a service. These state transitions are described formally so that a computer can understand them. So, a service specification means a set of all specifications for the service. ‘Execution of a specification’ means to execute a state transition described in the specification.
‘Triggering a specification’ means to initiate execution of the specification.

A concrete example of a seeming interaction between Call Forwarding service (CFV) and Calling Number Delivery service (CND), which is described in the international benchmark, is explained.

1) A specification of CFV

A typical specification of CFV is explained. Suppose that terminal A receives a dial tone (denoted by dialtone(A)), and terminal B has CFV activated and has registered terminal C as a forwarding terminal (denoted by cfv(B,C)). When terminal C is idle (denoted by idle(C)), if terminal A dials terminal B (denoted by dial(A,B)), the call from terminal A is forwarded to terminal C, then terminal A calls terminal C (denoted by calling(A,C)) (Figure 1).

Figure 1: A specification of CFV.

2) A specification of CND

A specification of CND is explained. Suppose terminal C has CND activated (Figure 1 A specification of CFV denoted by cnd(C)). When terminal A receives a dial tone and terminal C is idle, if terminal A dials terminal C (denoted by dial(A,C)), terminal A calls terminal C, and a telephone number of terminal A is displayed on terminal C (denoted by display(C,A)) (Figure 2).

Figure 2: A specification of CND.

3) Occurrence of a seeming interaction

Since displaying a telephone number is executed after it is determined that terminal C is called, a specification of CND is triggered later than that of CFV. In the conventional detection methods, since it was supposed that when an event, dial(A,B) occurs, only a specification which has dial(A,B) as an event was triggered, only CFV is triggered. Therefore, a telephone number of terminal A is not displayed on terminal C despite terminal A calls terminal C. Since this is an abnormal state, an interaction is detected.

However, taking into consideration the process flow in a real system, after execution of CFV, the call from terminal A is forwarded to terminal C. Then, if terminal C is idle, terminal A calls terminal C. In effect, it can be said that a call from terminal A reaches terminal C. Thus, the meaning of the event, which is a trigger for executing specifications after execution of a specification of CFV, should be deemed to be ‘a call from terminal A reaches terminal C’, that is, in this case, dial(A,C). After a specification of CFV is executed, a specification of CND which has dial(A,C) as an event is triggered. As a result, a telephone number of terminal A is displayed on terminal C and an interaction does not occur.

Therefore, the interaction between CFV and CND shown as an example is a seeming interaction.

3 TP MODEL

The minimum explanation of the TP model, which is necessary for this paper, is given. For more details please refer to (Shimokura, 2004).

3.1 Overview

The TP model is designed, independently from individual services, based on state transition diagrams. To realize independency, each system state in state transition diagrams for supplementary services is represented as one of abstracted states in state transition diagrams for the basic service (POTS). Thus, each state transition for supplementary services can be represented as one of state transitions between common states, Sc and Sn, abstracted from states of POTS (Figure 3).

Figure 3: Represented state transition.
timing points for specifications. Since timing points for individual specifications are determined according to TPs, the order of triggering specifications is clear.

State transitions for CFV and CND, described in section 2, are explained using Figure 3. Current states for CFV and CND, Sac and Sbc, can be represented as Sc. Next states for CFV and CND, San and Sbn, can be represented as Sn. TPs for a specification of CFV and a specification of CND are described as TPx, and TPy, respectively. Thus, it is clear that a specification of CFV is triggered before a specification of CND.

### 3.2 Triggering a Specification

(1) Conditions for triggering a Specification

Suppose that in a specification, ‘current state’ which is a service state before the state transition, ‘event’ which is a trigger for the state transition, ‘next state’ which is a service state after the state transition, the name of a TP where the specification is initiated, and a name of a TP or a state which the process flow after execution of the specification reaches, are described. If the process flow after execution of the specification does not reach one of TPs or states, the last term, a TP or a state, is not described. In the TP model, when an event Ei occurs and the process flow reaches one of TPs or states, the last term, a TP or a state, is not described. In the TP model, when an event Ei occurs and the process flow reaches one of TPs or states, TPi, specifications, which have the same event as Ei and the same trigger point as TPi, are initiated. Thus, for one event, more than one specification can be triggered.

Conditions for triggering a specification are given as follows:

Condition 1: The process flow for the event reaches a TP described in the specification.

Condition 2: An event described in the specification occurs.

Condition 3: A state described in the current state of the specification is the same as a service state when the process flow reaches a TP described in the specification.

(2) Triggering two specifications

In the TP model, when two specifications are given, each specification is initiated as follows:

(i) In case two specifications have the same TP.

When a process flow reaches a TP described in both specifications, firstly, a specification, which satisfies Condition 2 and Condition 3, described above in this section, is triggered. Where both specifications satisfy the conditions, either specification is triggered first. After execution of the first specification, if the process flow reaches a TP described in the second specification and the second specification satisfies the conditions, the second specification is also triggered.

(ii) Two specifications have different TPs which are set on the same state transition in the TP model.

When a process flow reaches a TP described in the specification triggered first, if the specification satisfies Condition 2 and Condition 3 described above in this section, it is triggered. After execution, if the process flow reaches a TP described in the other specification and the specification satisfies the conditions, the specification is also triggered.

(iii) Two specifications have different TPs which are set on different state transitions, respectively, in the TP model.

Since the temporal order of TPs cannot be determined, both specifications are triggered in the same way as case (i).

### 4 MEETING CONDITIONS FOR TRIGGERING: A PROBLEM AND ITS SOLUTION

#### 4.1 Problem

To detect interactions, it is judged whether two given specifications can be triggered or not.

In the TP model, when an event, Ei, occurs and the process flow reaches a TP, TPi, a specification which is triggered at the TPi has Ei as an event. But, as mentioned in section 2, there is a case where the meaning of an event is changed by execution of a specification. Therefore, in this case, after execution of the specification, a specification that has an event other than Ei may satisfy Condition 2 described in section 3.2. In this case, another specification that has Ei as an event does not satisfy the condition.

Therefore, to judge whether a specification which is executed after execution of the first specification, satisfies Condition 2 described in section 3.2 or not, it is necessarily to identify what an event means after execution of the first specification.
4.2 Solution

(1) Identification Method

The meaning of an event used in POTS is well known. But, the meaning of a new event used in supplementary services cannot be known beforehand. The meaning of an event commonly used in two supplementary services causes the problem in judging if Condition 2 described in section 3.2 is satisfied. But, most of those events are used in POTS. Therefore, in this paper, targeted events are restricted to those used in POTS. The meaning of those events is classified, and a method for identifying the meaning of events is discussed.

Because of space limitation, a method for identifying the meaning of dial(A,B) is discussed here.

For POTS, there is a specification that represents a state transition: when a service state is \{dialtone(A), idle(B)\}, if dial(A,B) occurs, the service state transits to calling(A,B). This specification can be taken as, when terminal B is idle, if a call from terminal A reaches terminal B, terminal A calls terminal B. Besides, only this specification represents a state transition to calling(A,B). That is, ‘when terminal B is idle, terminal A calls terminal B’ is a necessary and sufficient condition for the meaning of an event, which is a trigger for initiating this specification, to be that a call from terminal A reaches terminal B. Thus, if calling(A,B) is described in the next state of the specification, after execution of the specification, arguments X and Y of dial(X,Y) which means ‘a call from terminal X reaches terminal Y’, are A and B, respectively.

Thus, if calling(A,B) is described in the next state of a given specification s (in this case, s is called as s1) which has dial(X,Y) as an event, after execution of s, dial(X,Y) should be considered to be changed to dial(A,B).

An identification method in the case where calling(A,B) is not described in the next state of s is discussed in (2).

(2) calling(A,B) is not described in specification s

A method for identifying the meaning of an event in the case where calling(A,B) is not described in the next state of s, s2. A concrete example where calling(A,B) is not described in s2 is explained.

s2, which defines a state transition when terminal C registered as a forwarding terminal in CFV is not idle, is shown in Figure 4. Figure 4 is explained. Suppose terminal B has CFV activated and has registered terminal C as a forwarding terminal. When terminal C is not idle (denoted by not[idle(C)]), if terminal A dials terminal B, terminal A receives a busy tone (denoted by busy(A)). Thus, when s2 is executed, a call from terminal A reaches terminal C. Thus, the meaning of the event after execution of s2 should be regarded as not dial(A,B) but dial(A,C).

However, since calling(A,C) is not described in the next state of s2 shown in Figure 4, the identification method proposed in (1) above cannot identify the meaning of dial(X,Y).

But, in general, there are two cases for terminating terminal, idle and not idle, the service specification should have both specifications.

Thus, when calling(A,C) is not described in the next state of s2, by finding out another specification, s3, which has calling(A,C) in the next state, the meaning of the event can be identified as dial(A,C).

(3) An event identification method

For a method for identifying the arguments X and Y in dial(X,Y), after execution of specification s that has dial(A,B) as an event, discussion (1) and (2) mentioned above are summarized. When calling(P,Q) is described in the next state of s (s1), dial(X,Y) after execution of s1 is regarded as dial(P,Q). Here P and Q represent arbitrary terminals. When calling(P,Q) is not described in the next state of s (s2), identify another specification s3, that has dial(X,Y) as an event and calling(P,Q) is described in its next state in the service specification to which s2 belongs. If s3 is found, dial(X,Y) after execution of s2 is regarded as dial(P,Q). If s3 is not found, since arguments in dial(X,Y) cannot be identified, the arguments in dial(X,Y) after execution of s2 are regarded as unchanged. Consequently, there is a possibility that real interactions are not detected and/or seeming interactions are mis-detected. This possibility is evaluated in section 6.
Based on the discussion above, a method for identifying the arguments in an event after execution of specification s which has dial(A,B) as an event, is proposed as follows:

Step 1) If calling(P,Q) is described in the next state of specification s, go to Step 3.
Step 2) Search for another specification si, that has dial(X,Y) as an event and calling(P,Q) is described in its next state, in the service specification to which specification s belongs. If specification si is not found, go to Step 4.
Step 3) Identify the meaning of dial(X,Y) after execution of specification s as dial(P,Q), and end identification.
Step 4) Identify the meaning of dial(X,Y) after execution of specification s as dial(X,Y), and end identification.

5 DETECTION METHOD FOR INTERACTIONS

A new interaction detection algorithm, which is based on the TP model and adopts solutions described in section 4.2, is proposed. In interaction detection, for given two specifications depicted from two service specifications, respectively, non-determinacy interactions and semantic interactions, which means abnormality of a system state or a state transition after execution of two specifications (Ohta, 1994)(Ohta, 1998) are detected, as conventional detection methods.

5.1 Detection Scenario for Non-determinacy Interactions

A non-determinacy interaction occurs when the order of triggering two specifications cannot be determined. In the TP model, the order of triggering two specifications cannot be determined in the following cases: Both TPs and events described in each specification are the same, or events described in each specification are the same and the TPs of each specification are set on different state transitions in the TP model. So, in either case, a non-determinacy interaction is detected.

5.2 Detection Scenario for Semantic Interactions

(1) Checking conditions for a specification to be executed

Since a specification is described as a state transition, when two specifications that belong to different services are given, a specification, which satisfies all of the following conditions, should be executed.

(a) The process flow reaches a TP described in the specification.
(b) An event described in the specification is the same as one that actually occurs.
(c) The state described in the current state of the specification exists in the current service state of a compound.

The conventional detection methods (Ohta, 1994) (Yoneda, 2003) can be used for judging conditions (c). Judging conditions (b) can be made by using the identification method proposed in section 4.2. Therefore, the method for judging condition (a) is discussed.

If two specifications have the same TP, or different TPs which are set on different state transitions in the TP model, the both specifications are judged to satisfy condition (a).

In case that two specifications, sa and sb, have different TPs (TPa and TPb) that are set on the same state transition in the TP model, if TPa is set ahead TPb in the state transition in the TP model, specification sa is judged to satisfy condition (a). For sb, only if the process flow in the TP model reaches TPb after execution of specification sa, specification sb is judged to satisfy condition (a).

These judgments can be made by comparing a TP, described in specification sb, and a destination reached by a process flow after execution of specification sa, described in the specification sa.

(2) A detection scenario

A detection scenario of semantic interactions is proposed. When the given two specifications have different events, firstly, an event described in either of two specifications is supposed to occur, and detecting interactions is done. Then, suppose that the other event occurs, and detecting interactions takes place. The detection scenario is as follows:

Step 1) According to the triggering methods described in section 3.2, execute a specification that is triggered first.
Step 2) After execution of the specification, the event is changed if needed, according to the identification methods of events described in section 4.2 (3).
Step 3) Execute another specification according to the triggering methods described in section 3.2
and obtain a state of a compound service after a state transition.

Step 4) Judge whether each specification should be executed or not according to the method described in (1) above. If both specifications are judged to be executed, go to Step 6.

Step 5) If a state described in the current state of the specification, which is judged as not to be executed, does not exist in a state of a compound service after execution of the other specification, an interaction is detected. Go to Step 7.

Step 6) If each state described in the next states of each specification does not exist in a state of a compound service after execution of both specifications, an interaction is detected.

Step 7) If a state of a compound service after execution of specification/specifications violates either service constraint (Ohta, 1998), an interaction is detected.

6 EVALUATION

The event identification method proposed in section 4.2 and the new detection method for interactions proposed in section 5 were applied to specifications for 12 services, which are described in the international benchmark (Griffeth, 2000). In the international benchmark, 98 interactions are reported (Griffeth, 2000). But, among them, there are 22 interactions that do not actually occur because system states just before executing the specifications cannot actually exist. According to our investigation beforehand, it was confirmed that 39 interactions out of 76 interactions are seeming interactions.

For the identification method: in all cases for all pairs of 12 services, all events are correctly identified. Thus, the proposed identification method was confirmed to be reasonable.

For the detection method: in 39 seeming interactions (8 non-determinacy interactions and 31 semantic interactions) were avoided to be mis-detected, and 37 actual interactions were detected. Thus, the proposed detection method was confirmed to be effective.

7 CONCLUSION AND FUTURE WORK

To implement a detection system without mis-detecting seeming interactions, a method for identifying the meaning of an event was proposed. In addition, a new method for detecting interactions was proposed. The proposed method was applied to specifications of 12 services described in the international benchmark for interaction detection, and it was confirmed that the proposed methods were reasonable and effective.

For future work, an automatic detection system based on the proposed methods will be implemented and evaluated in more detail.

REFERENCES


REACTIVE, DISTRIBUTED AND AUTONOMIC COMPUTING
ASPECTS OF AS-TRM

E. Vassev, H. Kuang, O. Ormandjieva, J. Paquet
Department of Computer Science and Software Engineering, Concordia University,
EV 3.165, 1455 de Maisonneuve West, Montreal, Quebec, H3G 1M8, Canada
{vassev, kuang, ormandj, paquet}@cse.concordia.ca

Keywords: Autonomic Computing, Distributed Computing, Reactive Systems, Software Architecture.

Abstract: The main objective of this research is a rigorous investigation of an architectural approach for developing
and evolving reactive autonomic (self-managing) systems, and for continuous monitoring of their quality. In
this paper, we draw upon our research experience and the experience of other autonomic computing
researchers to discuss the main aspects of Autonomic Systems Timed Reactive Model (AS-TRM)
architecture and demonstrate its reactive, distributed and autonomic computing nature. To our knowledge,
ours is the first attempt to model reactive behavior in the autonomic systems.

1 INTRODUCTION

Autonomic computing is a new research area led by
IBM Corporation, which area concentrates on
making complex computing systems smarter and
easier to manage (Kephart, Chess, 2003; Horn, 2001,
Ganeck, Corbi, 2001). The main characteristic of
autonomic computing is self-management, i.e., self-
monitoring of its own use and quality in the face of
changing configurations and external conditions,
based on automatic problem-determination
algorithms. Many of autonomic systems concepts
imitate self-regulatory model of human autonomic
system; thus autonomic computer systems are
envisioned to combine the following seven
characteristics: self-configuring, self-healing, self-
optimizing self-protecting, self-defining,
contextually aware and anticipatory (Kephart, Chess,
2003, Horn, 2001). The first four characteristics
listed above are considered to be the core
characteristics of an autonomic computer system
(McCann, Huebscher, 2004).

However, according to our best knowledge,
autonomic computing technology has not been
applied to model and develop real-time reactive
systems, which systems have high demand for
autonomic computing technology to remove the
complexity of modeling and development. With
autonomic behavior, real-time reactive systems will
be more self-managed to themselves and more
adaptive to their environment.

Research Goals. The long-term research goals
for this project are: 1) modeling of distributed
autonomic reactive components along with their
relationships; 2) modeling of the qualitative
properties constraining systems’ behavior, such as
reliability. In order to achieve our research goals, we
need to: 1) develop an appropriate formal framework
for autonomic distributed real-time reactive systems
that leverages their modeling, development,
integration and maintenance, as well as for
continuous self-monitoring of their quality
formalism to support distributed autonomic
behavior, and 2) build corresponding architecture
along with communication mechanism to implement
distributed autonomic behavior. In this paper we
describe the architecture and the communication
mechanism for of Autonomic Systems Timed
Reactive Model (AS-TRM) by revealing its reactive,
distributed and autonomic computing nature.

Our Approach. We add reactiveness to the
autonomic components’ behavior thus allowing
them to communicate and synchronize with the
environment while fulfilling their tasks. The novelty
of our approach consists in combining the
advantages of both the formal representation of
reactive components in TROM formalism
(Achuthan, 1995), and the autonomy of components
in agent-oriented paradigm. Our objectives include
an extension of TROM as an Autonomic System Timed Reactive Model (AS-TRM) to include the specification of distributed reactive components along with their relationships, and the non-functional properties constraining systems’ behavior.

Fig. 1 illustrates the concept of the Reactive Autonomic System AS-TRM. To our knowledge, ours is the first attempt to model reactivity in autonomic systems.

2 AS-TRM ARCHITECTURE

This section provides the comprehensive conceptual view of the AS-TRM architecture; it is intended to capture and convey the significant architectural decisions, which serve as a foundation for the further design and implementation. We focus on the structural and the dynamic view, as well as on the specific characteristics of AS-TRM to discuss its reactive, distributed and autonomic aspects.

Our AS-TRM architecture builds upon extending the TROM formalism (Achuthan, 1995) for modeling reactive systems. Reactive systems are the computer systems that continuously react to their physical environment, continually sensing and responding to the environment, at the speed determined by the environment. Reactive autonomic systems have infinite behavior and must satisfy the following two important requirements for reactivityness:
- stimulus synchronization: the process is always able to react to stimulus from the environment;
- response synchronization: the time elapsed between a stimulus and its response is acceptable to the relative dynamics of the environment so that the environment is still receptive to the response.

The TROM formalism for developing real-time reactive systems is briefly introduced below.

2.1 TROM

Real-time reactive systems are some of the most complex systems, so the modeling and development of real-time reactive systems becomes very challenging and difficult work. The TROM formalism (Achuthan, 1995) for real-time reactive systems developed at Concordia University is a powerful tool for dealing with complexity issues in developing such systems. The TROM formalism is a three-tier formal model (Achuthan, 1995). This three-tier structure describes system configuration, reactive classes, and relative Abstract Data Types (see Fig. 2).
On the other hand, autonomic computing is the new research area, which focuses on developing complex computing system smarter and easier to manage. The goal of our work is to extend the current TROM formalism to AS-TRM to include the specification of distributed reactive autonomic components along with their relationships, and the qualitative properties constraining the system’s behavior.

2.2 AS-TRM

AS-TRM can be considered as TROM with extended autonomic behavior, and the autonomic functionalities are those creating the autonomic behavior. Autonomic functionalities can be implemented locally, using locally maintained measurements and knowledge. The autonomic behavior can be implemented among the members within a peer group through sharing measurements and knowledge of the group.

AS-TRM extends the TROM formal model by adding more tiers (see Fig. 3) and including the specifications for a time-reactive component (RC), an autonomic group of synchronously interacting RCs (ACG) and an autonomic system (AS) that consists of asynchronously communicating ACGs.

RC. This newly added tier encapsulates TROM objects (the TROM formalism’s second tier) into an AS-TRM reactive component. The synchronous interaction between the RCs allow for realization of a reactive task. The communication between RC and its upper tier ACG is realized through an interface and is asynchronous (see section 3.1).

ACG. AS-TRM Autonomic Group of RCs is a set of synchronously communicating RCs cooperating in fulfillment of a group task. Each ACG can accomplish a complete real-time reactive task independently. The self-monitoring behavior at the ACG tier and the asynchronous interaction between ACG and the RCs is realized by an ACG Manager (AGM). The responsibilities of an AGM include the following:
- Continuous monitoring of the ACG quality level required by the evolving nature of the peer group
- AGM monitors the behavior of the synchronously communicating RCs and analyzes the correctness of their functionalities according to the policies.
- AGM receives diagnostic messages from RCs and sends back treatment messages to them;
- AGM unplugs the broken RCs from their group and plugs them back when they are ready;
- AGM automates the initialization and maintenance according to evolving group configuration and changes in the run time;
- AGM encapsulates any data under control of this group and manages all data shared either between RCs or between the group and other groups.

The reactive behavior is modeled at the RC and ACG tiers. We model the environmental objects communicating with the system as RCs, and incorporate them into the ACG fulfilling the corresponding reactive task.

The self-healing and self-optimizing autonomic behavior can be implemented on peer group level as the following aspects:
- Automating backup of policies for the group;
- Knowledge and resource sharing within the group;
- Execution time optimization according to the empirical data.

AS. Autonomic System (AS) tier is abstracting a set of asynchronously communicating ACGs. The self-monitoring behavior and the asynchronous interaction between AS and the ACGs is realized by the Global Manager (GM). The responsibilities of the Global Manager (GM) include the following:
- Continuous monitoring of the AS quality level required enduring the safety of the autonomic system;
- GM verifies user access according to the security policies and different level privileges defined among GM, ACGs, RCs, and environment;
- GM monitors the behavior of the ACGs and analyzes whether they work correctly according to the policies;
- GM receives diagnostic messages from ACGs and sends back treatment messages to them;
- GM receives a request for updating the compositional rules for ACGs and synchronization axioms among RCs from the user, and forwards the updates to the AGMs.
The self-protecting, self-configuring, self-optimizing, and self-healing autonomic behavior can be implemented at the AS level as the following aspects:

- automating user access support;
- automating the configuration for users;
- knowledge and resource sharing within the system;
- automating the backup of the policies for the entire system.

**2.3 Characteristics of AS-TRM**

Below we summarize the autonomic characteristics of the AS-TRM architecture shown in Fig. 4 (based on a figure in (Bantz et al, 2003)) in addition to real-time and reactive those inherited from the TROM formalism (Achuthan, 1995):

- AS-TRM is self-managed: it can monitor its components (internal knowledge) and its environment (external knowledge) by checking the status from them, so that it can adapt to changes that may occur, which may be known changes or unexpected changes;
- AS-TRM is distributed: the components within AS-TRM can collaborate to complete a common real-time reactive task distributively;
- AS-TRM is proactive: it can initiate changes to the system;
- AS-TRM is evolving: a) the policies of each RC can be changed in the run time according to the changes of requirements; b) the composition rules of the RCs within corresponding peer group can be changed in the run time; c) the synchronization axioms among the RCs within corresponding peer group can be changed in the run time.

**2.4 Architecture of AS-TRM**

Our architectural goal is to capture the above-mentioned characteristics of AS-TRM. The architecture of AS-TRM (see Fig. 4) is based on the tiers of the AS-TRM formal model, and consists of Reactive Components (RCs), AS-TRM Component Group Manager (AGM), and Global Manager (GM), which are connected to each other at the local, peer group, and system levels.

At the peer group level, which is also the AS-TRM Autonomic Group of RCs (ACG) level, every AGM interacts and shares knowledge as well as information with its RCs; it receives information (policies) from its superior (Global Manager) and implements them with its own resources. The autonomic behavior at this level is a result of peer knowledge-sharing, getting local agreement, and acting locally on that knowledge. Fig. 5 is another architectural view of AS-TRM.

**ACG architecture.** An ACG consists of an AGM and a set of managed RCs. An AGM consists of a collection of intelligent agents which are responsible for the autonomic behavior of self-configuring, self-healing, self-optimizing, as well as self-protecting, and a replicator for replicating the states of the RCs within the ACG. The intelligent agents in the AGM can communicate another through the Autonomic Signal Channel. Each managed RC communicates its events and other measurements with the AGM. According to the input received from the RCs, the AGM makes the decisions based on the policies, facts, and rules (stored in the ACG repository) and communicates the instructions with corresponding RCs.

**Anatomy of GM and RC.** A GM consists of a set of intelligent agents which are responsible for the autonomic behavior of self-configuring, self-healing, self-optimizing, as well as self-protecting, and a replicator for replicating the states of the ACGs within the AS-TRM system. The intelligent agents in the GM can communicate each other through the Autonomic Signal Channel. Every ACG communicates its events and other measurements with the GM. According to the input received from the ACGs, the GM makes the decisions based on the policies, facts, and rules (stored in the AS repository) and communicates the instructions with corresponding ACGs.
3 AS-TRM COMMUNICATION SYSTEM

The AS-TRM Communication System (ACS) is an autonomous messaging system in the AS-TRM that exposes interfaces for both synchronous and asynchronous message-delivery services. By virtue of its architecture, the ACS is an application of the Demand Migration Framework (DMF) (Vassev, 2005) which extends the DMF architecture by adding new features for adaptation to the autonomic computing needs. The ACS architecture provides two means of communication among AS-TRM entities (RC, AGM and GM) – asynchronous and synchronous (see section 2.3). Asynchronous communication was inherited from DMF centralized message-persistent asynchronous communication, and synchronous communication is a variant of peer-to-peer communication (Vassev, 2005). The former takes place between the RCs and the AGMs, and between the AGMs and the GM. Peer-to-peer communication takes place between RCs. Fig. 6 depicts the layered architecture of the ACS derived from the DMF. The architectural ACS model consists of four layers – Message Space (MS), Message Space Proxies (MSPs), Transport Agents (TAs) and Peer-to-Peer Transport Agents (P2PTAs). While the MS, MSP and TA layers are derived directly from the DMF (Vassev, 2005), the P2PTA layer is an ACS extension that addresses synchronous communication issues. The ACS inherently relies on MS, MSPs and TAs to “form architecture applicable to asynchronous communication systems, where the messages are delivered in a demand-driven manner” (Vassev, 2005). The MS incorporates a persistent storage mechanism for all the messages exchanged asynchronously in the AS-TRM. The MS in turn incorporates an Object Query Language (OQL) (Emmerich, 2000) for querying the stored messages. On top of this, we have the MSP presentation layer. There is a single MS and multiple MSPs in the model, each MSP being associated with a TA.

The TAs (see the dark grey segments named TA in Fig. 6) form a migration layer (Vassev, 2005) for transporting messages asynchronously to and from the RCs, AGMs and the GM that adhere to the TAs’ interface. TAs are “independent stand-alone components able to carry objects over the machine boundaries” (Vassev, 2005). We use them to migrate messages from one node to another. The TAs provide a transparent form of migration. On top of the TA layer, we have the P2PTA layer (see the...
white segments wrapping the TA layer and bordered with a dashed circle), P2PTAs provide an alternative means of communication, which is synchronous point-to-point communication. The RCs use the P2PTAs for direct synchronous communication.

Figure 6: AS-TRM Communication System.

3.1 Messaging

The messages communicated via the ACS fall into two major groups – heartbeat and regular messages. The heartbeat messages are used for self-monitoring, and they provide a summary of the component state. The RCs send proactively and regularly their state to the associated AGM, and the AGMs send their state to the GM. The regular messages are the AS-TRM work and configuration messages. In addition, each AS-TRM message has a priority, recognizable by the transport agents. This helps messages with higher priority to be delivered first. From the functional perspective, the ACS addresses:
- asynchronous broadcasting;
- canceling messages;
- asynchronous and synchronous sending and receiving heartbeat and regular messages;
- asynchronous sending and receiving regular messages to and by a specified node.

3.2 Autonomic Features

The AS-TRM Communication System extends the DMF (Vassev, 2005) by implying some autonomic computing features like self-protection, self-optimization and self-configuration (see section 2). Some of the autonomic computing features are addressed by the DMF architecture (Vassev, 2005). The core components – MS and TAs, work in autonomous and independent mode. Hence, the ACS inherently consists of autonomous elements. In order to make the ACS’ components autonomic, we extend their autonomous architecture by adding to them a management unit (see Fig. 7). The management unit (MU) controls and monitors the associated ACS’ unit. Hence, each ACS’ component is a peer of autonomous units – a management unit and ACS work unit (WU). The MU performs control functions over the WU, which simply performs its work duty and reports proactively its state to the MU. The last can decide to shut down and/or restart the former if there is no state report received for an efficient amount of time.

Figure 7: ACS’ Components Architecture.

The ACS’ autonomic features are:

Self-protection. Only communication-trusted end-points are able to communicate via ACS. The ACS exposes an integrated security mechanism that prevents unauthorized access. This autonomic feature is inherited from the DMF architecture.

Self-configuration. The ACS is a distributed system with hot-plugging (Vassev, 2005) features. For example, the TAs are able to discover available MS and plug into the ACS. This autonomic feature is inherited from the DMF architecture.

Self-healing. The ACS’ components can be restarted by the embedded management unit. The ACS addresses at least one delivery semantics (Vassev, 2005), which prevents messages sent asynchronously to be lost. That allows the restarted component to continue from the point it stopped.

4 RELATED WORK

An autonomic system may contain many autonomic components that communicate and negotiate with each other and other types of resources within or outside of system boundaries. This is referred to as autonomic manager collaboration. The architectural concepts for autonomic systems are mainly based on
IBM Corporation’s blueprints and on the on-going research for autonomic computing conducted at the IBM’s laboratories (IBM, 2003, IBM 2004, IBM, 2005). The three blueprints are overviews of the basic concepts, constructs and behaviors for building autonomic capability into computer systems. The autonomic component architecture relies on the technique of feedback control optimization based on forecasting models, which technique facilitates the self-management features of an autonomic system.

In (Yuan-Shun, 2005), a new model-driven scheme for autonomic management is presented. This scheme can better allocate resources by using reliability models to predict and direct the distribution of monitoring efforts. If certain services or components are predicted to have high reliability at a particular time, then there is no need for intensive monitoring during that period, but those with low reliability require intensive monitoring.

Our research considerably differs from the related work in this area for the reason that we target the modeling of both reactiveness and autonomicity in distributed systems.

5 CONCLUSIONS AND FUTURE WORK

The research work reported in this paper is our first step towards developing a formal framework for developing distributed reactive autonomic components along with their relationships, and the qualitative properties such as reliability and safety constraining the behavior of the system. Particularly, it addresses: 1) the extension of the existing TROM formalism for modeling real-time reactive systems to AS-TRM formalism for supporting autonomic behavior; 2) the characteristics of AS-TRM for determining the requirement specification, design, and implementation of AS-TRM; and 3) the architecture and communication mechanism of AS-TRM for implementing autonomic as well as real-time reactive functionalities. This paper describes only the architecture aspects of AS-TRM, i.e. it does not deal with load balancing and efficiency aspects, as these are part of our future work on AS-TRM.

One of the most important aspects of autonomic systems is their self-management – a feature requiring formal mechanism for self-diagnosis of the AS-TRM system’s quality status. The evolving nature of the AS-TRM requires continuous monitoring of the quality levels to evaluate the risk of deploying a change on the configuration of the AS-TRM system, and to diagnose potential safety hazards in AS functionality. We are investigating means for achieving continuous quality assessment of the evolving AS-TRM. We intend to develop and analyze algorithms and negotiation protocols for conflicting quality requirements, and determine what bidding or negotiation algorithms are most effective.

REFERENCES

UNIFIED DESCRIPTION AND DISCOVERY OF P2P SERVICES

G. Athanasopoulos, A. Tsalgatidou, M. Pantazoglou
Dept. of Informatics and Telecommunications, National and Kapodistrian University of Athens
{gathanas,atsalga,michaelp}@di.uoa.gr

Keywords: Service Oriented Computing Peer-to-Peer services, Service Description, Service Discovery, Service Invocation.

Abstract: Our era has been marked by the emergence of the service oriented computing (SOC) paradigm. This new trend has reshaped the way distributed applications are built and has influenced current computing paradigms, such as p2p and grid computing. SOC’s main objective is to leverage interoperability among applications and systems; however, the emergence of various types of services such as web, grid and p2p services has raised several interoperability concerns among these services as well as within each of these service models. In order to surpass these incompatibilities, appropriate middleware and mechanisms need to be developed so as to provide the necessary layers of abstraction and a unified framework that will obscure a service user from the underlying details of each service platform. Yet, for the development of such middleware and mechanisms to be effective, appropriate conceptual models need to be constructed. Within this paper, we briefly present a generic service model which was constructed to facilitate the unified utilization of heterogeneous services, with emphasis on its properties for the modeling of p2p services. Moreover, we illustrate how this model was instantiated for the representation of JXTA services and present the service description and discovery mechanisms that were built upon it. We regard this generic service model as a first step in achieving interoperability between incompatible types of services.

1 INTRODUCTION

Contemporary application development trends have been highly influenced by the service oriented computing (SOC) paradigm (Papazoglou, 2003). This new software engineering trend has primarily affected the development of distributed systems. There have been many distinct definitions over what constitutes a service (Vogels, 2003), (Booth, 2004), (Foster, 2002). Nevertheless, all these definitions share some common properties and characteristics (Tsalgatidou, 2005). A service is usually regarded as a self-described software system offering a specific interface (that is described using a specific interface definition language) to its clients who can invoke it via messages over the Internet using predominant protocols and standards such as HTTP, XML, etc.

As it has been articulated by many researchers (Vogels, 2003), (Papazoglou, 2003), services leverage the interoperability among applications. SOC provides an infrastructure, which alleviates the interoperability concerns raised by the underlying platform, programming languages and operating systems used by software applications and systems.

Despite the original hype, the emergence of various types of services such as web, grid and peer-to-peer (p2p) types of services rendered the vision of interoperability elusive. Each of these types of services incorporates its own service model and attributes services with specific properties and characteristics. Heterogeneity is also prevailing among the underlying mechanisms supporting basic functionality such as the description, discovery and invocation of services.

Discrepancies and interoperability problems are not just inter-service type concerns. Such problems are also encountered within services of the same type. WS-I provided the Basic Interoperability Profile (WS-I, 2004) in order to overcome some of the interoperability concerns in the web service computing paradigm. Regarding p2p services, the existence of a plethora of p2p systems and frameworks, which abide by proprietary protocols and models, has aggravated the problem of p2p service interoperability. A comparison on some of the most well known p2p systems and frameworks
such as JXTA (JXTA), Gnutella (Gnutella) and Edutella (Edutella), reveals their diversities on the notion of p2p services. Each of these p2p frameworks and systems incorporates proprietary mechanisms and infrastructure for the description, discovery and invocation of p2p services and attributes p2p services with distinct characteristics and properties according to its own adopted service model.

All these diversities among the existing types of services hinder their widespread utilization and their seamless integration. In order to overcome the obstacles raised by the heterogeneity among the existing types of services one has to be based on a solid foundation. Such a foundation could be a conceptual model, which would provide the necessary basis for the development of appropriate abstractions and mechanisms that could facilitate service interoperability. This was one of the objectives of the SODIUM project (http://www.atc.gr/sodium). A Generic Service Model called GeSMO was developed within the SODIUM project and served as the basis for the development of a set of languages, middleware and tools for the unified discovery and composition of heterogeneous services such as web, grid and p2p services.

Within this paper we briefly illustrate this generic service model with emphasis on how this model was specialized for the representation of p2p services. Furthermore, we present how this model was instantiated for rendering JXTA services as well as the mechanisms that were developed for the description and discovery of JXTA services. This model has also served as the basis for the development of a p2p service invocation mechanism which will be the subject of another paper.

We have chosen to primarily focus on p2p services since this type of services present the widest range of diversities. Therefore, the application of our approach on this type of services will clearly illustrate its merits and the benefits gained by its usage.

Before proceeding we would like to give a definition on what constitutes a p2p service in order to dispel any misconceptions that may arise. As it has been stated in (Tsagkatiidou, 2005), p2p services can be classified either as coarse services providing high level business logic, such as file sharing and instant messaging or as elementary services providing basic p2p network functionality, such as discovery of nodes and resources, message routing and message exchange. As far as coarse p2p services are concerned, these may be defined as “the provision of resources or the execution of tasks of one or more (temporarily provider) peers on behalf of one or more (temporarily user) peers in a P2P network” (Gerke, 2003), whereas elementary p2p service may be defined as “services that support basic functionality in a P2P system, such as discovery of peers or resources, peer membership management, query formulation and routing, etc”.

The goal of this paper is to illustrate the provided generic service model and its extensions for supporting the representation of p2p services as well as how this generic service model was used for the development of appropriate mechanisms supporting the description and discovery of p2p services. In order to achieve its goals the paper is organized as follows: Section 2 illustrates the current state of the art on some of the existing p2p platforms and systems supporting the notion of p2p service. Consequently, sections 3 and 4 present respectively the generic service model and its instantiation for the JXTA service paradigm. Finally section 5 draws our conclusions and future plans on p2p services.

2 CURRENT STATE

A thorough look into some of the most prominent p2p platforms and systems unveils the set of discrepancies among these platforms regarding the properties and characteristics as well as the mechanisms used for supporting basic functionality of p2p services. The results of such an investigation on the JXTA (JXTA), Edutella (Edutella) and Gnutella (Gnutella) platforms are briefly presented below. A more detailed analysis of these problems can be found in (Tsagkatiidou, 2005)

– **Support for the Notion of Service**: JXTA is the only framework which inherently supports this notion and enables the development and provision of additional services. Edutella is a platform built on top of JXTA providing a basic set of services, described using web service protocols, which enable the exchange of resources among peers, annotated with meta-data (Nejdl, 2002). Gnutella, on the other hand, is a fully distributed protocol enabling the development of file sharing p2p applications. In contrast to JXTA and Edutella, Gnutella doesn’t accommodate the notion of service.

– **Syntactic Service Descriptions**: JXTA utilizes the advertisement construct which is a language neutral meta-data structure represented as an XML document. A service advertisement document conveys the necessary information
that enables a peer to invoke a preconfigured service. However, an advertisement document doesn’t convey information related to the interface, operations or messages that are exchanged by a service; all this information is hard-coded into the service and client applications. Edutella utilizes web service description protocols and standards such WSDL or DAML-S. These constructs convey the necessary information that can be used for the on-the-fly invocation of services. Gnutella, in contrast to JXTA and Edutella, doesn’t provide any form of service description document.

- **Supporting Mechanisms**: In terms of underlying mechanisms used for publishing, discovering and invoking p2p services, JXTA offers a set of services which facilitate the publishing and querying for available services as well as mechanisms for binding and exchanging messages with selected services. Edutella leverages the underlying mechanisms and services provided by JXTA for the discovery and invocation of the provided set of services. Gnutella on the other hand doesn’t accommodate such mechanisms supporting the publishing, discovery and invocation of p2p services.

- **Semantic Service Description**: Among the investigated p2p platforms only Edutella supports the semantic annotation of p2p services descriptions through the use of semantic description frameworks such as DAML-S. JXTA and Gnutella don’t accommodate semantic annotations to service descriptions.

All these incompatibilities among the investigated types of services render the seamless utilization of p2p services an arduous task. In order to overcome these shortcomings, developers have to provide customized solutions that facilitate the integration of such services. Nonetheless, such solutions are usually strongly bound to the underlying infrastructure as well as the specific application logic.

An appropriate solution that could be applied to this problem is to develop a generic service model which will specify all necessary constructs. This service model would provide the basis upon which middleware and tools could be built so as to accommodate the necessary abstractions. Such a model is presented in the following section.

### 3 GENERIC SERVICE MODEL

In this section, we briefly illustrate the structure and the concepts of a Generic Service Model (GeSMO), which can be used as a basis for the development of appropriate languages and middleware that address service interoperability. GeSMO was based on a thorough investigation of the current state of the art on web, grid and p2p services. Specifically with respect to the p2p services, GeSMO has been primarily influenced by the work in JXTA (JXTA) and Gnutella as the latter is one of the very few p2p networks that ardently support the notion of service.

The generic service model was constructed in such a way that it efficiently models all common characteristics of web, p2p and grid services, while at the same time provides for the modeling of the distinct characteristics per service type. The architecture that was selected for the development of the Generic Service Model (GeSMO) is a layered one consisting of a core layer, an extension layer built on top of the core layer and a number of layers orthogonal to the core layer and its extensions. More details on the GeSMO model can be found at (Tsalgatidou, 2005).

As expected, the fundamental element in GeSMO is the notion of service. Services are regarded as self-described software systems, which interact with their clients over the Internet through messages. As we can see in Figure 1, a service description facilitates its clients in identifying the messages that can be exchanged as well as where and how these messages should be sent.

![Figure 1: Basic Service Model.](image)

The syntactic elements of a service along with their structure are illustrated in Figure 2. According to Figure 2, a service provides one or more interfaces which consist of the operations that this service offers to its clients. An operation groups a set of messages that are exchanged among a service and its respective clients. Each message consists of a set of elements which adhere to specific data types.
3.1 P2P Service Model

The P2P Service Model extends the core concepts of the generic service model with additional elements that are needed for the description of p2p services. Although the extensions for the p2p service type were primarily influenced by the JXTA platform, they are still generic enough and thus applicable to other types of platforms.

According to Figure 3 the concepts that were added are that of P2P Service, Peer, Peer Group and PSDL (P2P Service Definition Language) description. A p2p service represents a service that is provided by a peer or a set of peers in a p2p network. A p2p service may be a coarse service or an elementary service according to the classifications provided in (Tsalgatidou, 2005).

A peer represents a node of a p2p network, which is able to communicate with other peers and provide them with services. A peer group represents the logical group of peers that may be formulated in a p2p network.

A PSDL description provides information on what a p2p service does and how it can be invoked. This concept was added so as to facilitate the description of p2p services with additional information elements. Hence, a PSDL document doesn’t aim at replacing the existing description constructs provided by the underlying platform, but rather to annotate them with additional elements that these constructs lack. Furthermore, PSDL descriptions provide the basis upon which mechanisms and middleware could be built so as to support the more efficient discovery and utilization of p2p services. More info about the PSDL description may be found in (Tsalgatidou, Athanasopoulos 2006)

The aforementioned concepts represent a basic set of elements that may be used for the description of p2p services. However, when it comes to modeling p2p services offered by specific platforms, such as JXTA or Edutella, additional elements or specializations of the aforementioned concepts might be needed. Such specializations for the JXTA platform as well as a description and discovery mechanism which were based upon this model are presented in the next section.
providers. Pipe Advertisements and Module Specification Advertisements are specializations of the advertisement construct which are used for publishing information about pipes and services respectively.

The specializations of the generic service model provided for representing the concepts of the JXTA platform are illustrated in Figure 4. An extended PSDL description for the JXTA platform describes the pipes that may be used for exchanging messages with a service and provides references to their respective Pipe Advertisements. A p2p service’s Module Specification Advertisement provides reference to the PSDL description document of that service. This association is accomplished through the use of an extension element (SURI element) that is inherently supported by a Module Specification Advertisement.

4.1 JXTA Service Description

A JXTA service description document such as the one presented in Figure 5 is an instantiation of the PSDL description concept that is illustrated in Figure 4. As it can be seen in Figure 5, PSDL is an extension of WSDL 1.1 (Christensen, 2001) which has the same abstract part as a WSDL document and an extended concrete part. The concrete part contains additional elements to accommodate the concepts described in Figure 3 and Figure 4. The WSDL 1.1 elements that were extended so as to accommodate the extra concepts presented in Figure 3 and Figure 4 are the Binding and Port elements. The Binding element has been extended via the jxta:binding sub-element (see Figure 5) which declares that this is a binding to a JXTA p2p service. Moreover, the jxta:binding element provides information on the type of interactions that might take place via this binding i.e. send, receive or send-receive interactions as well as the patterns of these interactions i.e. synchronous or asynchronous interactions. The Port element on the other hand has been extended with the jxta:provider, jxta:pipes and jxta:moduleAdv sub-elements. The jxta:provider element conveys information describing the provider of a p2p service which might be either a specific peer or a peer group (JXTA). The jxta:pipes element lists the pipes that support the exchange of messages with the service. For each pipe element that might be used for interacting with a service the pipe advertisement of that pipe as well as the interface accessible through that pipe are specified. Finally, jxta:moduleAdv element provides reference to the service’s Module Specification advertisement. A PSDL document provides all the necessary information for identifying the functionality a JXTA service offers as well as how it can be invoked.

4.2 JXTA Service Discovery

In this section, we will briefly describe the mechanism we implemented for the discovery of JXTA p2p services. Due to the lack of space, and the fact that this is not the main purpose of this paper, implementation details have been left out. As it is shown in Figure 6, the course of service discovery in general involves the following steps: a) Specify service requirements in the form of a query; b) Submit the query to the appropriate broker, registry or network, in order to retrieve service description.
advertisements; c) Match the contents of the query against those advertisements; d) Wrap matching services in a message and send it to the requester, as response to his/her request.

The USQL Engine (Tsalgatidou, Pantazoglou, 2005), our proposed service search engine prototype, implements the aforementioned generic service discovery mechanism. The USQL Engine is an extensible framework, capable of applying service discovery against heterogeneous service registries and networks, in a unified manner. As its name implies, it makes use of the Unified Service Query Language (USQL) for the formulation of incoming service requests and outgoing service responses. USQL abides by the principles of GeSMO, and allows requesters to express their requirements by making use of syntactic, semantic, as well as quality criteria. A detailed description of the language is given in (Tsalgatidou, 2006). Like GeSMO, both USQL and the USQL Engine have been developed within the context of the SODIUM project.

The USQL Engine retains its abstraction from heterogeneous service types and their related technologies by establishing a flexible plug-in based architecture. More specifically, the framework provides extension points for discovery agents and document handlers. The former are responsible for accessing and querying the various registries and networks by utilizing their existing discovery mechanisms, while the latter are responsible for parsing the various heterogeneous service advertisements hosted in those registries and networks.

In the case of JXTA service discovery, the USQL Engine has been extended with a JXTA Agent plug-in. This component utilizes the JXTA Discovery service, in order to access a peer group and retrieve the advertisements it contains. Provided that these advertisements contain a link to an external PSDL description document, the agent employs the PSDLHandler plug-in in order to parse these service descriptions and supply the USQL Matchmaker with input. The JXTA services that were found to meet the requester’s requirements are included in the resulting USQL response, which contains adequate information for their immediate invocation. The whole procedure is displayed in Figure 7, which may be perceived as an instantiation of the generic mechanism shown in Figure 6.

In the current implementation of the JXTA agent plug-in, the USQL Engine needs to become a member of a peer group, before applying service discovery. In a highly volatile environment, such as a p2p network, and given the asynchronous nature of advertisement retrieval in JXTA, this approach may not always be the best one. A more sophisticated approach would break the plug-in into two loosely coupled parts: one part which would be a peer in the peer group of interest, caching all service advertisements, and another part acting as a remote control of the peer, asking for service advertisements. Such an approach is within our plans for future work. Nevertheless, the USQL Engine framework is flexible enough to be able to accommodate both approaches, as well as other potential architectures. For instance, the USQL Engine being exposed as a JXTA service, it could be used as the default service discovery mechanism within the boundaries of peer groups and networks, thus significantly contributing to the JXTA infrastructure.

5 CONCLUSIONS

Service Oriented Computing has reformed the way distributed applications are built. P2P along with grid computing have been heavily affected by this new engineering trend. Although, grid computing ensued a straightforward approach in accommodating the service model through the (Foster, 2002) and (Foster, 2005) specifications, p2p computing has not ensued a similar approach.

The incompatibilities among the existing p2p platforms in supporting the notion of service are hindering the widespread utilization of p2p services as well as their interoperation with other types of
services. In order to surpass the difficulties raised by the discrepancies among the existing types of services as well as of the existing p2p service platforms a model describing the necessary concepts needs to be provided. Upon this model appropriate mechanisms and middleware can be built so as to provide the necessary abstractions.

In this paper, we presented a generic service model called GeSMO, which was developed within the SODIUM (http://www.atc.gr/sodium) project for facilitating the modeling of web, grid and p2p services. The extensions that were provided for the description of p2p services as well as an instantiation of the p2p service model for the JXTA platform were also presented. The extensions provided for JXTA were the basis for the development of appropriate mechanisms supporting the description and discovery (Tsalgatidou, Pantazoglou, 2006) of JXTA services.

Despite that currently the presented p2p service model was only instantiated for the description of JXTA services, its concepts are generic enough so as to enable the modeling of other types of p2p services such as Edutella services or Gnutella services. Furthermore, although we have presented mechanisms which facilitate the description and discovery of JXTA services, additional mechanisms can be provided so as to enable the invocation of these services.

Our future plans include the instantiation of this model for other types of p2p service platforms e.g. Edutella, and the development of mechanisms which will enable the invocation of p2p services. Further future plans include the extension of the generic service model so as to accommodate other emerging types of services such as sensor services.

ACKNOWLEDGEMENTS

This work has been partially supported by the Special Account of Research Funds of the National and Kapodistrian University of Athens under contract 70/4/S829 and by the European Commission under contract IST-FP6-004559 for the SODIUM project.

REFERENCES


JXTA, Retrieved from http://www.jxta.org/


Keywords: Software Engineering, Software Maintenance, Maintenance Efforts Classification, Statistical Process Control.

Abstract: Imprecise effort estimations are a well known problem of software project management that frequently leads to the setting of unrealistic deadlines. The estimations are even less precise when the development of new product releases is mixed with the maintenance of older versions of the system. Software engineering measurement should assess the development process and discover problems occurring into it. However, there are evidences indicating a low success rate of measurement programs mainly because they are not able to extract knowledge and present it in a form that is easy understandable for developers and managers. They are also not able to suggest corrective actions basing on the collected metric data. In our work we propose an approach for classifying time efforts into maintenance categories, and propose the usage of maintenance charts for controlling the development process and warning about scheduling problems. Identifying scheduling problems as soon as possible will allow managers to plan effective corrective actions and still cope with the planned release deadlines even if unpredicted development problems occur.

1 INTRODUCTION

Effort estimation is known to be one of the most challenging problems of software project management. Recent studies show that only about 25% of software projects are successfully completed in time and in budget (Liu et al., 2003). Effort estimations are more imprecise when maintenance activities of older system versions are run in parallel with development of new product releases. When making release plans, project managers need to take into account the efforts required for implementing new functionality for the next release as well as the efforts required for correcting old system defects and new defects discovered into productive systems and the available human resources, too. In the world of software engineering that is so complex and so immaterial there are a lot of events that brake these plans. In reality there are no ideal cases where each part of a project is completed exactly as scheduled. Being short before or behind schedule is not a problem as long as the process is under statistical control and within predicted risk limits. One of the most important problems of software project management is that without having appropriate warning mechanisms, managers discover too late schedule overruns, wrong estimations and software quality problems and it is too late to correct and minimize their effect (Florac and Carleton, 1999). In order to be able to deliver projects in time, budget and with a high level of quality, project managers need to be early warned about the risks associated with a project that runs out of control (Liu et al., 2003).

Software Engineering Measurement (SEM) is a key practice in high maturity organizations. The 4’th Capability Maturity Model Integration (CMMI) level, also known as qualitatively managed level, defines key practices for quality management and process measurement and analysis. Companies situated on this maturity level start to use quantitative measurement and use statistical process control for improving the quality and increasing the efficacy of their processes. Unfortunately, under relatively restricted budgets conditions, small and medium software companies are not able to effectively introduce these practices into their development process.

---

*This work is carried out with financial support from the EU, the Austrian Federal Government and the State of Carinthia in the Intereg IIIA project Software Cluster South Tyrol - Carinthia

2 see http://www.sei.cmu.edu/cmmi/ for reference
cess management is not a business goal in these companies, and they are not ready to pay the relatively high costs associated to software process measurement and analysis. Goethert and Hayes present a set of experiences from implementing measurement programs indicating that measurement programs have a success rate below 20% (Goethert and Hayes, 2001). The measurement programs usually fail because the collected metrics are found to be irrelevant or not well understood by key players, expensive and cumbersome. Also no actions on the numbers are suggested, and some of the collected metrics are perceived to be unfair and the developers manifest against their usage (Brown and Goldenson, 2004), (Goethert and Hayes, 2001).

In this paper we try to counteract the presented management and measurement problems by proposing an approach based on:

- Collection of time efforts and their classification into maintenance categories.
- Building of maintenance charts
- Warning about development and scheduling problems. The collection of time efforts and software metrics can be automated by employing tools like Prom (Sillitti et al., 2003) or HackyStat (Johnson et al., 2003). In section 3.2 we present three models used for classifying the maintenance efforts. In this way we present the results in an easy understandable form to managers and developers. In this paper we support the hypothesis that the maintenance charts and the warning mechanism presented in section 3.3 are valuable solutions for software process assessment, helping the managers to easily interpret the evolution in time of maintenance efforts and to find the sources of scheduling problems.

2 RELATED WORK

Software measurement is a research topic since many years, and still continues to be an open research field due to the continuous evolution of software technologies, paradigms and project management techniques. In the followings we reference a selection of related work in the areas of software metrics, software maintenance, and artificial intelligence techniques applied in software engineering.

The software process improvement (SPI) is a hot topic in software industry, which bases itself on the collection of product and process metrics. In order to be effective SPI must employ tools that automatically collect metrics with low costs high quality (e.g. manually collected data are error prone and influenced by human judgement). Hackystat (Johnson et al., 2003) and Prom (Sillitti et al., 2003) are so called SPI tools of third generation, that facilitate the collection of product (software) and process metrics (time efforts spent for in designing and developing software projects).

Because of the "legacy crisis", described by Seacord et al. in (Seacord et al., 2003), the measurement and estimation of software maintenance efforts gained special attention starting with '80s. Studies referenced by Seacord et.al. show that the most life cycle costs of information systems occur after the first software release. Another studies published in the 80’s showed that on average the corrective efforts take about 20% of the total maintenance efforts while adaptive efforts take about 25%, and the most part of 50% is directed to perfective category (Lientz and Swanson, 1980). Usually, preventive efforts are not greater than 5% of the total maintenance efforts (Seacord et al., 2003). More recent studies from environments involving newer technologies confirm the same distribution of maintenance efforts (Vliet, 2000), even in the case of web applications (Lee and Jefferson, 2005).

Generally, the information used in these reports is extracted from change logs, issue tracking systems and/or version control systems, which is manually collected and usually incomplete and error prone (Graves and Mockus, 1998; Kemerer and Slaughter, 1999; Zanker and Gordea, 2006). From our knowledge the work presented in this paper is the first attempt of classifying efforts in maintenance categories basing on automatic collected time information.

In the last years, different artificial intelligence techniques were employed for extracting knowledge out of the metric data and for learning models that assess different software engineering tasks like: predictions and estimation of software size & quality, development and maintenance efforts and costs (Zhang and Tsai, 2003). Similar to our approach Liu et al. present a warning system for early detection of scheduling and budgeting problems, as well as low quality risks, based on software metrics and rules extracted with a fuzzy inference engine (Liu et al., 2003). Different from Liu’s work we focus our attention on the evolution in time of development and maintenance efforts, and reasoning on maintenance charts. An analysis of software maintenance data using bayesian networks, decision trees and expert networks is presented in (Reformat and Wu, 2003).

Some software metrics are strongly correlated with each other, therefore using all available metrics to infer a decision model does not necessary improve the resulting model. Contrary, there are cases when complex models based on large sets of variables provide inferior prediction accuracy than alternative models based on smaller sets of variables (Thwin and Quah, 2005), (Khosgoftaar et al., 2003).
3 MONITORING AND CONTROLLING MAINTENANCE EFFORTS

In the development of almost all information systems there is a high pressure to release the first working version of the system as soon as possible making a compromise between the time to market and the quality of software products. Afterwards, the systems enter into a maintenance process with enhancement/modernization cycles and periodical new version releases (Seacord et al., 2003). In many cases, when working on a new release, the activities related to the implementation of new functionality are mixed with the ones related to the correction of defects found in previous releases. While the first category of efforts are typically payed by the customer, the second type of costs are covered by maintenance fees. Under this assumptions it is very important to measure and control the distribution of the development efforts over different maintenance activities.

3.1 Maintenance Categories

Taking into consideration the reasons of software changes, the maintenance efforts were classified by Swanson and Lientz into 4 categories (Lientz and Swanson, 1980): perfective, corrective, preventive and adaptive.

The perfective maintenance (PeM) typically consists of activities related to implementation of new system functionality, which usually take more time to be completed than other development activities. When enhancing system functionality new classes are added into the system and new methods as well (in existing and/or in the new classes). Under these conditions the value of all metrics, representing structural or complexity changes is increasing, and the share of time efforts spent for these activities are relatively high.

The corrective maintenance (CM) deals mainly with the elimination of system defects (also called bugs in software development communities). It affects existing artifacts, by changing parts of the source code that cause system misbehavior, which typically means correction or even re-implementation of existing algorithms. Usually, this kind of maintenance modifies the complexity and the size of existing artifacts without changing their structure too much, but in some cases the structure is also significantly affected. For example, there are cases when old pieces are deleted because they are not used anymore, or cases when the algorithms don’t consider all possible combinations of the input variables. In the last case it is required to treat new special cases by implementing new classes or methods. Depending on the severity and the nature of the corrected defects, the tasks associated to this maintenance activities may be completed in larger or smaller time intervals.

Preventive Maintenance (PrM) gained special attention in the last 20 years, when the demand for high quality was constantly increasing. Preventive maintenance activities have the goal to improve the quality of the source code and correct those parts of the code that are suspected to introduce future system defects. In this category are included the so called "code reviews", and also the agile practices like implementation of test cases, refactorings.

Preventive Maintenance - implementation of test cases (PrM_T). Since agile practitioners emphasized the test driven development, many companies started to adopt unit testing as an important component of their development process. It aims at verifying software’s correct functionality and early identification of system defects. For implementing unit tests, java developers extend the functionality of JUNIT\(^3\) library and implement project specific test cases. Identification of test cases in the source code can be done basing on the naming conventions (test classes include the "Test" prefix, or suffix in their names), basing on the class inheritance tree (test cases are subclasses of JUnit’s TestClass case) and physical location of the source files (test cases are kept apart from project’s source code, they are usually placed in folders that are exclusively dedicated to unit tests). Similar to perfective maintenance, this type of maintenance activities creates new artifacts, changing the structure of the source code. Since these artifacts are quite simple and small the efforts invested for their creation are relatively low.

Preventive Maintenance - Refactoring (PrM_R). Refactorings are changes of the internal structure of source code that improve its modularity, readability and understandability without changing its observable behaviour (Fowler, 1999). The source code refactorings have the goal of reducing the amount of duplicated code and improving its reusability. Refactorings may occur at different levels of the project structure: method, class, package, architecture. The effect of these activities are important changes in the structure of the code and a reduction of its size and complexity. When the refactored methods are not reused (e.g. refactoring is done to support future reuse, or just to simplify the algorithms), the overall size of the artifacts is preserved (no lines of code are added, or deleted, they are just restructured). Because of automatic support provided by development environments, simple refactorings require less effort in comparison with other development activities.

Adaptive Maintenance. The efforts required to modify software systems in order to be able to work

\(^3\)see http://www.junit.org for reference
in new environments (e.g., new operating system, new hardware, new databases etc.) are considered to be adaptive maintenance. We focus our research and experiments on systems developed in Java, which is a platform independent programming language. In this context this maintenance category is expected encompass insignificant amounts of efforts, and it is out of the scope of this paper’s work.

Source code comprehension (SCC). Source code comprehension is a software engineering and maintenance activity necessary to facilitate reuse, inspection, maintenance, reverse engineering, reengineering, migration, and extension of existing software systems. Typical for this activity is the fact that the programmers spend time just for visualizing source code, without making any change into it. The efforts invested in these activities need to be redistributed over the other maintenance categories. A simple solution for this problem is the distribution of these efforts according to the proportion of each maintenance category.

3.2 Classification Methodology

When searching for a robust classifier for maintenance efforts, we evaluated the performance of several models based on domain knowledge, induced decision rules and probabilistic models. The classification itself is done by analyzing the time evolution of a set of software metrics: Chidamber-Kemerer metrics, Halsted’s metrics and McCabe’s cyclomatic complexity. Additional two boolean variables, as well as the collected efforts themselves complete the list of classifiers’ input. The boolean variables represent the results of the tests indicating whether a given code fragment is part of a test class (TC), or whether it was created as a result of source code restructuring (OA).

The OA test is based on the concept proposed by Godfrey et al. (Godfrey and Zou, 2005) and on other approaches that aim at identifying structural changes in source code based on software metrics (Kontogiannis, 1997; Germain and Robillard, 2005). A detailed description of the other above mentioned metrics can be found in (Norman Fenton, 1997).

3.2.1 Knowledge-based Approach

The knowledge-based approach captures the domain heuristics in the classification table (see Table 1) and transforms into a decision rules representation (see Table 2). The given heuristics represent rules of thumb such as: If high amount of effort is spent for heavily changing the structure of a code fragment without increasing its size and without reusing code from other parts of the product, then the effort should be classified as corrective maintenance (compare the row marked with an asterisk in Table 1). In fact the heuristics formalize the discussion of the different maintenance categories in the previous sections.

The metrics in classification model are grouped into two classes. The STR group indicates structural changes and uses the following metrics: Number of methods, Number of classes and Depth of inheritance tree. Furthermore, the size and complexity metrics are grouped in the variable SIZE including: Lines of code, Response for a class, Fan-out, Cyclomatic complexity, as well as Halstead’s volume. EFF stands for the time effort associated with a given activity, while OA and TC signify decision variables on origin analysis and test classes.

Table 1: Classification Table.

<table>
<thead>
<tr>
<th>STR</th>
<th>SIZE</th>
<th>EFF</th>
<th>TC</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>SCC</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PrM_T</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CM</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PrM_T</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>CM</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>PrM_T</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>PrM_T</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PrM_T</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>CM</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>PrM_T</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>PrM_T</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PrM_T</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PrM_T</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>PrM_T</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PrM_T</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>PrM_T</td>
</tr>
</tbody>
</table>

For TC the value 1 signifies that the maintenance activity that is currently analyzed is related to the implementation/modification of test methods. OA equals 1 indicates that - within the course of the given activity - code fragments were extracted from the body of other methods. Small values of time effort (EFF) are marked with 0, while higher ones are marked with 1. Changes in the source code that increase its size (SIZE) or its structural complexity (STR) are indicated with the value 1 in the corresponding columns, while the value 0 means no change or a decrease of related metrics values. We inferred the following classification rules by using the Matlab statistical toolbox (Zanker and Gordea, 2006):

When analyzing the extracted categorization rules, we can observe that all efforts related to refactoring or correcting of test classes are classified with PrM_T.
Table 2: Classification rules.

\[
\begin{align*}
Pr_M &= TC \\
PeM &= \neg TC \land STR \land SIZE \\
CSS &= \neg TC \land \neg STR \land \neg SIZE \\
Pr_{MR} &= (\neg TC \land STR \land \neg SIZE) \land (\neg TC \land STR \\
&\land \neg SIZE \land EFF \land \neg OA)
\end{align*}
\]

Instead of \(CM\) or \(Pr_{MR}\). This definition is consistent with developers’ view, that considers only modifications of source code that implements a system behavior as perfective or corrective maintenance. A more detailed discussion regarding the expert heuristics basing on concrete source code examples is presented in (Zanker and Gordea, 2006).

3.2.2 Machine Learning Approaches

Machine learning algorithms are widely used for extracting knowledge out of empirically collected data sets. The most popular algorithms are based on decision trees or decision rules as well as on probabilistic models or neural networks.

**Decision Rules.** Decision trees, decision tables and decision rules are related knowledge representation technologies. Decision trees are classification schemes that consist of a set of subsequent Boolean tests that end up with leaves indicating the item’s category. All paths in the tree starting with the root and ending with one of the leaves can be expressed in the form of “IF (condition) THEN category” rules, where a condition is a conjunction of tests in the path. This is in fact the decision table representation of the decision tree. The description for a class can be described as a disjunction of all rules in a decision table identifying the given category, where its representation is generally known as disjunctive normal form, or decision rule.

Basically, there are two approaches for learning decision rules from a given data set. The top-down approach is also used for learning decision trees, and consists of an algorithm that recursively splits the data set until all sets contain elements belonging to only one category. The bottom-up rule induction approach is a two step algorithm. In the initial phase a decision table is constructed by collecting all individual instances from a data set. The second step of the algorithm builds generalized rules written in a more compact form by heuristically searching for the single best rule for each class that covers all its cases. A good comparison of available algorithms used for learning decision trees and decision rules can be found in (Apte and Weiss, 1997).

**Bayesian Networks.** Bayesian Networks, also known under the names of causal or belief networks are directed acyclic graphs (DAG) used to represent probability distributions in a graphical manner. Each node in the Bayesian network represents a probability variable, and has an associated probability distribution table used to compute class probabilities for any given instance (see Figure 1). An edge between two nodes of the network represents the direct influence of a variable representing the parent node to the accessor’s node variable. If there is no edge between two nodes of the network, their variables are considered to be conditionally independent (\(P(A/B)=1\)).

The computation of class probabilities is based on the Bayesian theorem:

\[
P(B/A) = \frac{P(A,B)}{P(A)} = \frac{P(A/B) \ast P(B)}{P(A)}
\]

where \(P(A)\) and \(P(B)\) are the probabilities of event A, and B respectively, and \(P(B/A), P(A/B)\) are the conditional probabilities, of event B given A, and of event A given B, respectively.

Given the fact that Bayesian networks are acyclic graphs, they can be ordered such that for each node all of its accessors get a smaller index. In this case, considering the conditional independence assumption between the parents and the accessors of network nodes, the chain rule in the probability theory can be represented as (H. Witten and Frank, 2000):

\[
P(a_1,a_2,a_3,\ldots,a_n) = \prod_{i=1}^{n} P[a_i/a_{i-1},\ldots,a_1]
\]

where \(a_i\) are networks nodes.

Learning and selecting the best Bayesian classifier from labeled data sets is a challenging problem. Many different approaches were proposed, most of them exploiting particularities of the Bayesian network and optimizing the learned models for particular probability distributions. A general and robust algorithm based on the minimum description length (MDL) principle is presented by Lam & Bacchus in (Lam and Bacchus, 1994).
3.3 Maintenance Charts

Building maintenance charts. The development efforts are not the only indicator of problems occurring in software projects, but all these problems will be reflected in maintenance charts generating instabilities or out-of-control situations. The statistical process control theory defines well established algorithms for building different types of control charts (range, average, etc.). The control limits are computed basing on previously collected data and using the concept of three sigma allowed variance. These charts can be built only after some amounts of empirical data are collected, and they are static models that will need to be changed in different phases of development process. Therefore we propose a model for building maintenance charts basing on initial effort estimations that will define the center line of each chart (CL). The upper control limit (UCL) and the lower control limit (LCL) are computed using the risk interval taken into consideration in project planning.

In Figure 2 we present an example of a maintenance chart that monitors and controls the evolution in time of perfective, corrective and preventive efforts. The maintenance efforts are not uniformly distributed over the whole development period of a new release. In the initial phase (Phase I) important amounts of efforts are allocated for designing new modules and for correcting defects of the last release (corrective maintenance), activities that are usually associated with refactorings and unit testing activities (preventive maintenance). In this phase, feature implementation activities postponed from previous releases are implemented too. In the second phase (Phase II) the most efforts are allocated for implementing new functionality into the system (perfective maintenance), while the last period before release (Phase III) is reserved for testing and correcting the found defects. In case of experienced development teams these tasks are associated with unit testing and refactorings. Given this distribution in time of the maintenance efforts, the maintenance charts are created as a combination of normal (simple average and range control charts) and moving average charts. The average distribution of the development efforts over maintenance categories indicates a healthy development process (∼65% Perfective maintenance, ∼25% Corrective maintenance and ∼10% Preventive maintenance).

Warning about development and scheduling problems. Four tests that are effective in detecting local unusual patterns in control charts are presented in (Florac and Carleton, 1999). These tests analyze the distribution of successive points in the control charts over the three sigma interval around the center line. They are used to identify if the process runs out of control (the variables overpass the control limit) or when the system lososes its stability or calibration (the variables doesn’t have a random variation around the center line). We adopted two of these tests that together with trend analysis are able to uncover process instabilities and warn about impending scheduling problems. The first test checks the existence of four or more points on the same side of the center line. A positive result of this test shows process instability and warns that the process may soon run out of control (see situation 1 in Figure 2).

The second test identifies the cases when the processes are out of control like in situation 2 of Figure 2 when the first point that overpasses the control limits is found. Apparently, less efforts invested in preventive actions are not an indicator of scheduling overruns since the planned functionality is still implemented into the system. Anyway, in this situation the managers must be aware that the last implemented source code was not enough tested and its quality was not verified. In other words, this source code may be buggy and software quality problems may occur in the near future.

In the third case (situation 3) the process runs completely out of control. The trend analysis shows a constant increase of corrective and a decrease of perfective maintenance efforts. Because of the deterioration of the source code quality implemented in the last period of time, it is harder to implement new functionality and more defects need to be corrected. In order to be able to make the release at the planned date it is absolutely mandatory to make corrections in the schedule. In order to bring the project back on track, the manager may decide to postpone the implementation of some system features for the next release and to reallocate these resources for improving the qual-
ity of the source code and for correcting more system defects.

4 EVALUATION OF CLASSIFICATION MODELS

The purpose of our evaluation was to compare the classification performance of the presented techniques. For the empirical evaluation we collected time efforts and software metrics from a student project over one calendar month. During the evaluation period the students were implementing their graduation project having the size about several tens of thousands lines of code. At the end of each day the students were asked to manually classify the collected efforts into the corresponding maintenance categories. This information was collected into a database consisting of 2155 events. Each event registered the entity that was edited, the date and its time effort, as well as the manually inserted maintenance classification of the developers.

Due to daily annotation of the experimental data set by developers, we assume the manual classification to be correct. Now we evaluate the accuracy of the expert’s set of heuristics and the learned classification models on the data set.

We compare expert heuristics (expert) with the three learning techniques (Bayes Net and induced decision rules). The classification accuracy was determined by cross-validating on 50% of the data set. Using the developers classification as relevance set, the classification performance of each algorithm was evaluated using the precision and recall metrics, which are the standard evaluation metrics used in information retrieval. Precision is defined as the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved. In our case, given the maintenance category X, the precision measures the ratio of events identically categorized by classification algorithm and software developers as belonging to category X from the total number of records selected by classification algorithm into category X. Similar to this, the recall is the ratio of events identically categorized by classification algorithm and software developers as belonging to category X, out of the total number of records classified by developers into category X. These two measures are inversely related and the best classification algorithms are those that present highest values for both metrics.

As can be seen in Figure 3 the expert model and learned decision rules provide the best results. Due to their classification rule that is realized with a single variable (TC), the PRMT efforts are identified with 100% accuracy by these two algorithms. Contrastingly, the probabilistic model identifies PRMT with high precision but introduces false positives (recall < 1). Perfective maintenance can also be predicted with a good precision (> 0.87) by all three models, but the expert heuristics are the only algorithm that do not introduce many false positives (i.e. also high recall). All algorithms have problems to correctly predict the source code comprehension activities (about 60-70% precision) and the machine learning models have problems to classify corrective maintenance, too. With a precision around 85% and a recall of about 77%, the expert model classifies corrective maintenance efforts with a reasonable accuracy.

Concluding, the expert model provides the highest prediction accuracy and is able to correctly classify about 83% of all events. The prediction accuracy remains stable over time. Using absolute effort numbers about 86% of total effort has been correctly classified.

5 CONCLUSIONS

Being able to deliver product releases at the planned deadlines is extremely important in software industry, especially for companies that work under con-
tract. Monitoring the progress and keeping the development process under control ensures the success of a project. However, there are many sources that produce development and scheduling problems in software projects. In this paper we presented an approach for warning about development and scheduling problems based on maintenance charts. Three types of tests inspired from statistical process control theory are used to identify events indicating instabilities or processes that get out from statistical control. An experiment evaluating the performance of different models used for classifying efforts into maintenance categories is presented. For this experiment we used an empirical data set collected from the development of a student project. The evaluation showed that a classifier based on expert heuristics outperformed a machine learning algorithm due to a higher stability versus false leads and noise. Future work will focus on the implementation of the presented concepts for assessing the management of commercial projects and further experiences can be acquired.

REFERENCES


ADVANCES ON TESTING SAFETY-CRITICAL SOFTWARE
Goal-driven Approach, Prototype-tool and Comparative Evaluation

Guido Pennella, Christian Di Biagio
MBDA-Italy SpA, Via Tiburtina, Roma, Italy
<guido.pennella, christian.di-biagio>@mbda.it

Gianfranco Pesce
Centro di Calcolo e Documentazione, Università degli Studi di Roma “Tor Vergata”, Via O. Raimondo, Roma, Italy
pesce@ccd.uniroma2.it

Giovanni Cantone
Dip. di Informatica, Sistemi e Produzione, Università degli Studi di Roma “Tor Vergata”, Via O. Raimondo, Roma, Italy
cantone@uniroma2.it

Keywords: Software engineering, Distributed and parallel systems, Hard Real-time Systems, Performance-measurement Tools.

Abstract: The reference company for this paper – a multination organization, Italian branch, that works in the domain of safety-critical systems – evaluated the major tools, which the market provides for testing safety-critical software, as not sufficiently featured for their quality improvement goals. Consequently, in order to investigate the space of possible solutions, if any, the company’s Research Lab. started an academic cooperation, which led to share knowledge and eventually establish a common research team. Once we had transformed those goals in detailed technical requirements, and evaluated that it was possible to realize them conveniently in a tool, we passed to analyze, construct, and eventually utilize in field the prototype “Software Test Framework”. This tool allows non-intrusive measurements on different hard-soft targets of a distributed system running under one or more Unix standard OS, e.g. LynxOS, AIX, Solaris, and Linux. The tool acquires and graphically displays the real-time flow of data, so enabling users to verify and validate software products, diagnose and resolve emerging performance problems quickly, and enact regression testing. This paper reports on the characteristics of Software Test Framework, its architecture, and results from a case study. Based on comparison of results with previous tools, we can say that Software Test Framework is leading to a new concept of tool for the domain of safety-critical software.

1 INTRODUCTION

This paper expands on a previous work (Di Biagio, 2006b), which investigated the major available technologies for testing hard real-time software. The main result of that study was the characterization of those technologies from the point of view of a certain company – a multination organization, Italian branch, which works in the domain of safety-critical systems.

Based on the results from that study, the company’s management evaluated the major tools that the market provides for testing safety-critical software, as not sufficiently featured for their quality improvement goals. Consequently, in order to investigate the space of possible solutions, if any, the company’s Research Lab. was allowed to start an academic cooperation, in the aim of sharing knowledge and eventually establish a common project and research team. This paper reports on some results and a product that derived from such an experience.

Let us briefly present the context of real-time performance testing, remanding to a technical report for further details (Di Biagio, 2006a). The usage of a monitor is strongly recommended for the test of performance of hard real-time systems (Tsai, 1995) and quality assurance of new digitalized safety-critical systems (EPRI, 1994). A monitor is a system able to observing and analyzing behaviors shown by another, in case remote, system (a.k.a.: the “target”).
comparing the actual states of the target with expected ones – as produced by the same monitor performing in the role of “oracle” (Weyuker, 1982) – or reporting on system failures – as detected by the same monitor performing in the role of “supervisor” (Simser, 1996) – respectively. In safety-critical applications, the system should be monitored by another safety system to ensure continued correct behavior. To achieve these goals, observed behaviors must be quickly accepted or rejected; this task is quite difficult to enact when complex real-time systems are involved, and the requested response time is not in the range of human capabilities. Additionally, software practitioners cannot diagnose, troubleshoot, and resolve every component affecting a critical software performance by using just manual methods.

The goal (Basili, 1994) of the present paper is concerned with the purpose of measuring system test performances. The focus is on measurement of CPU and memory loads, performance monitoring of distributed heterogeneous processes and their threads, intrusiveness, and other key attributes. The point of view consists in the reference organization practitioners. The context is the development of critical software. In particular, we want to proceed by: (i) expressing the reference company need of testing safety-critical software in terms of conveniently feasible features and capabilities; (ii) developing a new software tool that meet those needs; (iii) Characterizing that tool, comparing it with other testing tools, accepting it by a case study, and eventually (iv) accrediting the tool in field and continually improving it, based on feedback from practitioners (Cantone, 2000).

In the remaining of the present paper, Section 2 transforms the reference organization’s needs and goals in required testing features. Section 3 presents the philosophy, architecture, and functionalities of Software Test Framework (STFW), a new prototype tool, which is based on those features. Section 4 shows results from a case study, which involved the STFW. Section 5 briefly compares STFW with major professional tools that the market provides. Section 6 presents some conclusions and points to future research.

2 TESTING FEATURES

There is not enough room here to report on the interview-based requirement elicitation process that we enacted with the customer stakeholders (the reference company’s software practitioners and project managers). Anyway, based on the expected use cases and the resulting requirements, a list of testing features (F) follows, which, in our view, characterizes a software test framework and is able to satisfy the needs that the reference organization expressed. Each of the shown features is augmented with the F’s: (i) function or capability, (ii) measurement model applied (in round brackets), (iii) relative importance or weight, as expressed by the involved stakeholders [in square brackets] (values are not shown; see Section 5).

- F1 Heterogeneous targets monitoring (N|Y, heterogeneous target types) [w1].
- F2 Average CPU percentage used during data acquisition on a target system. CPU and memory (see F3) occupancies are calculated under their maximum load, i.e. when all possible data are required for acquisition, and the acquisition interval is the one suggested by the tool producer, respectively (%) [w2].
- F3 Memory occupancy on a target system (MB) [w3].
- F4 Persistent data repository and management (N|Y) [w4].
- F5 Tailor the test system to suit special user needs or purposes (N|Y) [w5].
- F6 Un-intrusiveness (Intrusiveness: time for data acquisition in seconds) [w6].
- F7 Distributed targets monitoring. TCP/IP over Ethernet (N|Y) [w7].
- F8 Plug-in architecture (N|Y) [w8].
- F9 System CPU (idle and used) percentage measurement (N|Y, %) [w9].
- F10 System memory load (free and occupied) measurement (N|Y, MB) [w10].
- F11 Process CPU (idle and used) percentage measurement (N|Y, %) [w11].
- F12 Process memory load (free and occupied) measurement (N|Y, MB) [w12].
- F13 Thread CPU (idle and used) percentage measurement (N|Y, %) [w13].
- F14 Thread memory load (free and occupied) measurement (N|Y, MB) [w14].
- F15 Support multi platform for all the major operative systems (N | (Y, Checkbox for LynxOS, Solaris, AIX, Linux, POSIX etc., respectively)) [w15].
- F16 Allow regression testing (N|Y) [w16]
- F17 Utilize software sensors (N|Y) [w17]. Cost (0*$) [w18].

3 SOFTWARE TEST FRAMEWORK

Software Test Framework is a complex analysis tool that deals with capturing resource occupation data of one or more target systems.
3.1 Architecture

In order to introduce minimal perturbation in the target system, STFW is developed for performing flexibly non-intrusive as-accurate-as-possible measurements. These results are achieved by employing a distributed architecture, which works on different computers in such way that only the measurements operations are performed on the target system, leaving the most complex elaborations and activities, such as the graphical plot, to other computers. Figure 1 shows the architecture of STFW. STFW is build-up by three macro-units:

- **Target**: it resides on each target machine and is responsible of the execution of the measurements and the optimization of the sensor. Target is build-up by two sub-units:
  - Test Manager (TM): its task is to opportunely tailor the Sensor.
  - Sensor: its task is to acquire information.
- **Analysis System**: it does not reside on a target computer but on a different machine. The Analysis System is responsible of the analysis, interpretation and visualization, both in real and in deferred time of data, which the instances of Sensor send. The Analysis System is build-up by three sub-units:
  - Data Manager: it is responsible for the interpretation of information sent by Sensor. The Data Manager also forwards the Data Plotter.
  - Data Plotter: is able to graphically plot data that Data Manager sends.
  - GUI (Graphical User Interface): sends Test Manager the information to acquire, as specified by the user.
- **Repository**: it historicizes test related data. The Repository does not reside on a target computer but on a different machine.

The most interesting features and capabilities of STFW are:

- STFW supports regression test
- STFW supports data repository
- STFW supports threads monitoring
- Sensor is a tailor-made software
- Sensor is not intrusive
- Acquisitions from different targets are synchronous in the same conversation (Anderson, 1983).

3.2 Usage

STFW is very easy to use. After the installation of the required software on Target, Analysis System, and Repository, a user is able to start with tests of any kind and proceed step by step. In the first step, the user chooses the information needed (concerning CPU, memory, and so on), the duration of the whole test, and the sampling interval by means of the STFW graphical user interface. In the second step, the user sets the IP addresses of the Target and Repository sub-systems. Now, the user is allowed to start the test. After a small time (1 – 20 sec), in which the Test Manager (TM) configures Sensor to acquire only the specified information (Sensor loads only the needed modules), data plotting is started on the user screen and, in parallel, the repository is populated.

The user, during the first step, can load and launch a historicized test: as result, the user is allowed to compare two different tests in the same plot, the historicized one, and the other one in running. Moreover, once a test is finished, the user can choose graphical or numerical presentation of results; plots are presented for each acquisition time.

3.3 Regression Test

STFW provides EXnee, which is an integrated and enhanced version of Xnee. This is a free software tool, which is able to record and playback all events used by the X Server. So, each time a user moves the mouse or digits a button on the keyboard, Xnee records these events and is then able to reproduce all the related actions. In this way, Xnee is able to replicate in the system the effects of all the activities performed by the user in the same temporal sequence.
After a session of events is recorded, an STFW user can reproduce that session every time it is needed. Let us consider, for instance, a user, who starts the execution of a (critical) software, and then begins to interact with it. Of course, if the user makes decision to change that software, Xnee allows that user (and all the authorized colleagues) to start replication of all those interactions. Once that such a replication has been started, Xnee is able to proceed autonomously (the physical presence of user is no more requested) by replicating events of user-system interactions and identifying differences in behaviors, if any, due to the injection of software changes since the last build (regression test).

3.4 Tailoring

Concerning the consumer side, STFW is configurable to the different operational environments. In order to allow the (static) specialization of STFW to the particular operational environment, some parameters are specified for the framework (i.e., operating system, process monitoring, thread monitoring etc.); parameters are easily handled, due the STFW modular structure.

3.5 Intrusiveness

Intrusiveness represents for a software application the OS load. It is complementary to, and can be quantified in terms of, CPU percentage and amount of memory used by the application software itself in situation of maximum performance.

STFW is able to guaranty CPU occupancy under 1%, while acquire data with a minimal interval of 1 second. Let us note that major tools suggest acquiring data on the target system with sampling period not less than 3 or 10 seconds, respectively. Such a STFW advantage derives from its tailoring features (see Section 3.4) and the system architecture of the Target module.

3.6 Parallelism, Synchronization and Heterogeneity

Based on the architecture of our tool (see Section 3.1), STFW supports data acquisition in parallel from different heterogeneous targets. On a target machine, a test is build-up by a configuration phase and a subsequent conversation phase for data acquisition. When all the Sensors have been configured, they synchronize on the reception of a start message. Following the reception of this message, all Sensors start to acquire their data and finally sending those data to the consumer.

Let us note that, in order to compare consistent data, starting and completing synchronously acquisitions from different targets is an essential requirement. Because the end of a communication time-window is in the control of the consumer, it is enough to start (multi-point to point) communications at the “same” time, as STFW actually does (notice that latencies - as introduced both by the TCP/IP over Ethernet, and the OS scheduler – are negligible in common test environments, compared to sampling interval).

3.7 Data Repository

The whole information, as each Sensor acquires, is stored in a relational data base (DB). In order to keep intrusiveness in control, the DB is installed on the computer that hosts the Analysis System, or any other machine but different from the ones where Sensors are installed.

Storing data in a repository is useful because it allows reusing previous test cases, analyzing previous results, and comparing such previous results with those generated by running test cases.

3.8 Process and Thread Monitoring

STFW is able to acquire information about processes and threads, as in the followings:

- PID: Process Identifier
- TID: Thread Identifier
- PPID: Parent PID
- S: Status; can be Ready, Running or Waiting
- MO: Memory occupancy; is the sum of the amount of memory allocated for the stack, the executable file, and data.
- CPUO: CPU occupancy; is the percentage of CPU used.

TID does not apply to processes. In case of threads, MO evaluates the stack size (a thread shares text and data with its parent process).

4 CASE STUDY

Let us present results from a case study, where we compared in real-time the behaviors of two applications running on two Single Board Computer (SBC). Monitored attributes were the system’s target CPU occupancy, and the full information associated to the execution of two processes, Ubench 2.0 and Sensor, respectively. The Ubench job consists in computing senseless mathematical operations for 3 minutes, and then, in the successive 3 minutes, performing senseless memory allocation and de-
allocations (Ubench, 2006). The job of Sensor consists in auto-monitoring activities.

We conducted the case study in the reference company’s industrial environment, built-up by three calculus nodes, as in the followings: (1) Thales – Vmpc6a Single Board Computer (SBC) with Lynx OS, (2) Concurrent - Intel SBC with Linux Red Hat Enterprise, and (3) x86 PC with Windows XP. Those nodes are one to each other connected through an Ethernet LAN.

Each SBC was arranged to perform in the role of target system, and had its own Test Manager and Sensor installed. The Windows PC was arranged to perform in the role of consumer, and hosted the graphical console. Hence, we proceeded with the case study by starting Test Managers (i.e. writing “./testman” on the bash consoles) and the GUI (i.e. double clicking the exe file in the PC window). Following the start of the GUI, we passed to configure the targets by entering “CPU”, “Ubench” and “Sensor” and then pressing the OK button. When the Sensors were compiled, installed and ready to send data, we pressed the START button and then two plotting windows appeared on the PC screen, which showed the required information only.

Figure 2 shows an instance of process-monitoring windows in STFW.

5 COMPARATIVE ANALYSIS

In Table 1 we compare STFW with three major professional tools (Di Biagio, 2006a), (Di Biagio, 2006a).

Table 1 shows the limits of commercial measuring tool with respect to STFW.

In fact, for all the attributes of the evaluation model less the memory occupancy on a target (F3), STFW shows the same or better values than the other tools.

Consequently, in order to compare those technologies, we do not need to weight those attributes and develop a synthetic indicator: the advantage of STFW would persist to any practical set of weights chosen.

Anyway, the reader should notice that STFW is just a prototype (but in its second internal release).

While Table 1 is auto-explicative in terms of comparative analysis, let us use this opportunity to present some further considerations.

In our view, the measuring tools available are “heavy” both for data-producers and data-consumers. They admit the worst configuration only, so that they acquire all possible data.

Conversely, the installation of all their data-acquisition modules is permanently requested. As a result, consumers receive data that they never requested. As a further result, the intrusiveness is unnecessary high; in fact, it is proportional to the amount of data acquired. Instead, STFW is a framework, fully tailor-made: tailoring introduces improvements both on the producer side (unnecessary modules are not loaded), and the consumer side (only explicitly requested data is processed and represented to the consumer).

With respect to other monitoring technology, two turning points make STFW a new concept tool. Concerning the target machine, STFW reduces the occupancy of the system resources in term of memory and CPU percentage occupied, because only user-required data is acquired (no overload of the system resources), and memory allocation is
minimal (only the requested modules are loaded, which correspond to the requested data). Concerning
the consumer side, this is allowed to choose a-priori
the data to acquire, so not having to discriminate a
posteriori among all the received information for the
interesting data.

6 CONCLUSION AND FUTURE
WORK

We have presented the philosophy, architecture and
features of a new tool, STFW, for testing time-
behavior of safety-critical systems, and briefly
compared that tool with major system performance
measurement tools, as available from the market, to
the best of our knowledge. STFW resulted to be
much more supportive than other tools for our
reference professional engineers. The most
important features, which make STFW really a
competitive tool, are: (i) Tailor-made non-intrusive
data sensing; (ii) Synchronous conversations for
acquiring state information from distributed targets;
(iii) Repository of test cases for reuse, and their
results for comparative analysis; (iv) Thread
monitoring, (v) Ability to perform regression test.

Thanks to STFW, each product can be validate
and verified in real-time by monitoring and
comparing results from different tests, and
reproducing complete scenarios build-up by
different machines. Next step will be to extend
STFW to VxWorks™ (VxWorks, 2006), the
worldwide known OS for real-time system, and the
most utilized for the control of automata.

REFERENCES

Anderson T. and Knight J.C., A Framework for Software
Question Metric Approach, Encyclopedia of Software
Di Biagio C., Pennella G., and Cantone G., Comparing
Tools for Testing Critical Software. The Case Study of
“Software Framework 2.0”, TR 20060426.1, MBDA
Italy, 2006.
Di Biagio C., Pennella G., Lomartire A., and Cantone G.,
An Introduction to Characterization of Monitors for
Testing Safety-Critical Software, Proc. of ICSOFT 06
(these Proceedings), Setubal, 2006.
Cantone, G., and Donzelli P., Production and Maintenance of Goal-oriented Measurement Models,
International Journal of Software Engineering &
Knowledge Engineering, World Scientific Publishing
EPRI, Handbook for verification and validation of digital
systems, Vol.1: Summary, EPRI TR103291, Vol.1,
1994.
IEEE, IEEE/EIA 12207.0-1996 Industry Implementation
(ISO/IEC 12207) Standard for Information
Technology Software Life Cycle Processes, in
Isaksen U., Bowen J. P., and Nissanke N., System and
Lilja D. J., Measuring Computer Performance, Ed.
QUEST SPOTLIGHT™ http://wm.quest.com/library/
docs/spotlightwindows/SpotlightWindows.pdf (last
access, March 2006).
Simser D. and R.E. Seviora, Supervision of Real-Time
Systems Using Optimistic Path Prediction and
Rollbacks, Procs. Int’l Symp. Software Reliability
SOLARIS PERFORMANCE METER™ 2.0.0
http://docsun.cites.uiuc.edu/sun_docs/C/solaris_9/SUNWa
be/CDEUG/p125.html (last access, March 2006).
TOP™ - William LeFebvre’s
http://www.uwsg.iu.edu/UAU/system/top.html (last
access, March 2006).
Tsai J.J., Yang S.J., Monitoring and Debugging of
Distributed Real-Time Systems, J.J. Tsai and S.J.
Ubench 2.0™ ,
http://www.phystec.com/download/ubench.html (last
access, March 2006).
Weyuker E.J., On Testing Non-Testable Programs, The
VxWorks, http://www.windriver.com (last access, April
2006).
A SYSTEMATIC REVIEW MEASUREMENT IN SOFTWARE ENGINEERING

State-of-the-art in Measures

Oswaldo Gómez, Hanna Oktaba
Institute of Investigations in Applied Mathematics and Systems, Autonomous National University of Mexico UNAM
Scholar Circuit University City, Coyoacán 04510, Mexico City, Mexico
oswaldog@uxmcc2.iimas.unam.mx, ho@hp.fciencias.unam.mx

Mario Piattini, Félix García
Alarcos Research Group, Department of Computer Science, University of Castilla-La Mancha
Paseo de la Universidad/4,13071, Ciudad Real, Spain
Mario.Piattini@uclm.es, Felix.Garcia@uclm.es

Keywords: Software Measurement, Measure, Systematic Review.

Abstract: The present work provides a summary of the state of art in software measures by means of a systematic review on the current literature. Nowadays, many companies need to answer the following questions: How to measure?, When to measure and What to measure?. There have been a lot of efforts made to attempt to answer these questions, and this has resulted in a large amount of data what is sometimes confusing and unclear information. This needs to be properly processed and classified in order to provide a better overview of the current situation. We have used a Measurement Software Ontology to classify and put the amount of data in this field in order. We have also analyzed the results of the systematic review, to show the trends in the software measurement field and the software process on which the measurement efforts have focused. It has allowed us to discover what parts of the process are not supported enough by measurements, to thus motivate future research in those areas.

1 INTRODUCTION

It is a well-known fact nowadays that software measurement helps us to better understand, evaluate, and control the products, processes, and software projects from the perspective of evaluating, tracking, forecasting, controlling and understanding (Ebert et al., 2004). On the one hand, software measurement allows organizations to know, compare and improve their software quality, performance, and processes. On the other hand, software measurement helps organizations to estimate and predict software characteristics to support better decisions (Pfleeger, 1997; Florac et al., 1999). As a consequence, software measures are proving to be very effective for understanding and improving software development and maintenance projects (Briand et al., 1996), showing problematic areas in system quality and institutionalizing software process improvement.

It should also be noted that there is a large amount of studies in software measurement, which makes it very easy to lose information and to get confused. For this reason, it is important to follow a specific, strict, and very well defined method for searching in the current literature. If we take a look at software measurement, we realize that it is considered to be among the youngest disciplines, and it is currently in the phase in which terminology, principles, and methods are still being defined and consolidated (Briand, 2002). This means that there is not a general agreement about the exact definitions of the main concepts related to measurement. In addition, no single standard contains a complete vision of software measurements (Garcia et al., 2004).

With respect to the issues identified above, this article carries out a systematic review with a predefined search strategy, in order to summarize and classify the current and ongoing efforts in this field. The systematic review has been conducted according to the (Kitchenham et al 2004) proposal, which is very suitable for looking for information
about measures on different sources in a disciplined and systematic way. Hence, Systematic review allows us to recognize, evaluate and do even more; it helps us to identify issues for planning future investigation and provides us with information about the consistency of our results (Travassos et al., 2005). We chose systematic review because of its scientific methodology that goes one step further than a simple overview.

The goal of this work is to find and clarify the answers to three different questions: What to measure, when to measure and how to measure. This is achieved by analyzing from the results of the literature review, the following issues: proportion of measured entities; measured attributes; validated measurement; measurement focus; and measurement in life cycle software process.

This paper is organized as follows. After this introduction; an overview of the systematic review process is given. In the third section, the way in which the systematic review has been carried out on the software measurement field is explained. Then, an analysis of the results is provided. Finally, the conclusions and future work are dealt with.

2 SYSTEMATIC REVIEWS

It is often recognized in Software Engineering that different research studies are generally fragmented and limited, not properly integrated, and without agreed standards (Kitcheham et al., 2004). In order to avoid those problems we chose the systematic review to carry out this investigation on software measures. Systematic review aims to present a fair evaluation of a research topic by using trustworthy, rigorous and auditable methodology, along with a very well defined strategy that allows the completeness of the research to be executed (in this case on software measures). Furthermore, systematic literature review is a formal and methodological process that allows us to identify, evaluate, and interpret all existing studies that are related to our investigation on software measures based in this case on a research question, but it could be also based on topic area, or phenomenon of interest. This is done in such a way that it helps us to summarize the evidence that is currently available concerning a treatment or technology. It also serves to identify any gaps in the current research, and thus suggest areas for further investigations, and finally provide a framework/background to position new research activities appropriately.

The review provides us with the necessary information to properly address the software measures, by mapping the measure field, finding the relevant data, ideas, techniques and their correlation with our investigation. Besides, it can support the planning for a new piece of research. Moreover, with this systematic literature review we can integrate empirical investigation, in order to find out generalizations. We do this by establishing specific objectives to create critical analysis. An overview of the systematic review is provided in the next subsection.

2.1 The Systematic Review Process

In order to address and present a fair evaluation of a research topic, the systematic review is composed of the following phases:

Review Planning Phase: Here the investigation’s goals are established. The Review Protocol, which is the most important item in this phase, is generated. First and foremost, this protocol defines the research question and the methods that will be executed in the review. In a broad manner, this phase involves the following, summarized, activities, defined by (Travassos et al., 2005):

Question Formulation: This activity is considered to be among the most important in the systematic review process. Here the investigation targets must be defined by focusing the question and by establishing its Quality and Amplitude.

Source Selection: Primary studies from sources are selected here, by defining a source selection criterion, setting the studies’ languages, identifying and selecting the sources after an assessment of them and checking references.

Study Selection: It describes the process and criteria for the evaluation and selection of studies.

Review Execution phase: This phase involves identification, selection and evaluation of primary studies, based on the inclusion and exclusion criteria defined in the Review Protocol. It is composed of the following steps, in summary form:

Selection Execution: This section aims to register the selection process for primary studies by evaluating them with quality criteria.

Information Extraction: Once primary studies are selected, the relevant data must be extracted by following an Information Inclusion and Exclusion Criteria Definition, by defining Data Extraction Forms, and by resolving divergences among reviewers.

Result Analysis: In this phase all the information from the different studies is analyzed. This phase
involves the next step: **Result Summarization**, which presents the data resulting from the collected studies by doing **Calculus Statistical**, **Results Tables**, **Sensitivity Analysis**, **Plotting**, which will lead to the **Conclusion** and **Final Comments**.

The whole process must be stored and the planning and the execution have to guarantee that the research can be done. It is worth mentioning here that the **Review Protocol** must be evaluated by experts. Finally, many of the activities of the review process involve iteration to refine the process, and therefore they are not necessarily sequential.

In the next section, we describe how the review process, which was designed as appropriate to our research goals, was performed

### 3 SYSTEMATIC REVIEW ABOUT SOFTWARE MEASURES

First of all, it must be emphasized that this paper is an attempt to answer this fundamental question: What are the most current and useful measures in the literature? Since our whole protocol was produced around this question, this is the main step in our **Review Planning Phase**. Moreover, we hope that this work will be useful for project managers and software developers. The defined strategy was the following: first and foremost, the large collection of paper in current literature about software measurements was examined. Due to the great diversity of topics in this field, and with the aim of clarifying and summarizing them in the best way possible, we used the classifications of concepts defined in the Software Measurement Ontology proposed by (García et al., 2004). This ontology aims at contributing to the harmonization of the different software measurement proposals and standards, by providing a coherent set of common concepts used in software measurement.

In order to do the research we built the following combinations of search strings:

“**(measure OR metric OR quality OR quantitative) AND (process OR engineering OR maintenance OR improvement OR Software testing OR development)**”.

All the possible combinations with these words were tested in the following web search engines: **ACM Digital Library**, **Search IEEE magazines**, **Wiley Interscience**, and **Science@Direct**.

The results obtained on the web engines are shown in Table 1.

<table>
<thead>
<tr>
<th>Sources</th>
<th>Search Results</th>
<th>Reviewed</th>
<th>Accepted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Science@Direct</td>
<td>3569</td>
<td>78</td>
<td>10</td>
</tr>
<tr>
<td>ACM</td>
<td>950</td>
<td>85</td>
<td>28</td>
</tr>
<tr>
<td>IEEE</td>
<td>3740</td>
<td>111</td>
<td>32</td>
</tr>
<tr>
<td>Wiley</td>
<td>653</td>
<td>20</td>
<td>8</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>8912</strong></td>
<td><strong>294</strong></td>
<td><strong>78</strong></td>
</tr>
</tbody>
</table>

As we can see in Table 1, search engines provided us with 8912 papers. Nevertheless, it should be pointed out that only 78 were accepted, which represents about 1 % of the total articles, hardly even that. It is apparent that many articles were rejected. This is so because if a more limited search had been carried out, it would certainly have been true that we would have started with fewer results from the search engines, but at the same time we would have lost important articles. Therefore, a very less restrictive search was defined: as a result of this, we obtained too many articles, of which very few were considered apt. Furthermore, we have discarded those measures that were outside the scope of our model. We have also discarded measures that did not provide any relevant information, as well as repeated measures proposed by more than one author so that each measure is included only once. Hence, our attention focused on papers where keywords and titles included the research strings. These strings were also searched for in the whole document by some search engines.

Regarding the execution phase of the systematic review, the selection and evaluation of information was initiated using the terms of the inclusion and exclusion criteria defined in the review protocol. These criteria established that selected studies were in English and that all of them showed current, useful software measurements, basically only studies about measures for software development, software project administration and maintenance were selected. All papers had to satisfy our quality criteria and in this sense it is important to point out that all the searched-for sources are serious and that the quality of their papers is guaranteed. Moreover the search engines were validated by experts. For this reason, our quality criteria also trusted in the quality of the sources.

Once the papers were selected, the information was extracted by means of an extraction template for objective results which includes study name, author, institution, journal, date, methodology, results, problems and subjective results which includes information through authors, general impressions
and abstractions, according to the proposal provided by Trassvasos et al., (2005); in particular, the aims of this template are to store the results of the execution phase process by extracting, not only the objective information, but also the subjective information from each article analyzed.

Finally, in the results analysis phase we analyzed the measures in order to show, among other aspects, the information about attributes, the entities measured and their characteristics, the amount of measures in a specific attribute or entity, etc. This phase is described in more detail in the following section.

4 RESULT ANALYSIS

The measures extracted from the studies were summarized in terms of the Software Measurement Ontology, which helped us to find out what kinds of measures exist. More specifically, this ontology supported us in defining a template by categorizing the measures in the following three different ways: What to measure? How to measure? And When to measure?

Consequently, in order to summarize the existing measures, the ISO 15504, CMM, and CMMI establish a quality background for the improvement of maturity levels defining the Project, Process and Product as the kind of entities that can be measured. That is why we extracted attribute and sub-attributes (Fenton and Pfleeger, 1997) measured of these entities, from the articles reviewed and classified them into internal or external. With this part of the analysis we try to answer the question: What to measure? How to measure? And When to measure?

Once the measurements were collected and stored in our template table, we analyzed the amount of measures which have been defined for the Process, Project and Product kind of entities. As we can see in Figure 1, the most measured kind of entity is the project, and the entities whose measurement has been less supported by the current literature are the project and process. The reason is that measuring product is easier than measuring process and project, in which we usually find ambiguous definition of attributes. For products, quality and technical attributes are very well defined because quality has been strongly focused on product. Finally, measurements on product entities help to measure process and project ones.

Next, we shall look at another closely-related issue, which is the amount of measured attributes. Figure 2 shows the proportion of measure attributes according to our analysis of the accepted papers. As Figure 2 shows, size is one of the most measured attributes. The point is that the size is a base measure, not only needed in most of the derived measures, but the size measure is also easier to obtain because it focuses on one of the most “tangible” attributes which is the source code. Moreover, size has very well defined scales, units and methods of measurement like functions Points (FP) (IFPUG, 2004); therefore it is very difficult to get confused with size measurements. Furthermore, cost estimation is derived from size and the overall productivity, and finally the schedule is based on the size and cost estimates (Ebert et al., 2004). Hence size is used on most of control measures in a software project. The arguments set out here lead to an explanation of why size has one of the highest values in Figure 2.

In order to show in a in a better way the information displayed in Figure 2, Table 4 show the attributes order by the most measured.

In connection with the most measured attributes, the complexity attribute is used in different contexts, for example: source code complexity, Design complexity, UML Diagrams complexity, Architecture complexity, etc. Hence it can be seen that complexity has gathered many measurements from its different applications. If we take a look at Figure 2 in greater detail, it should be pointed out that attributes like Activity, Role, Work products and Accuracy are the least measured. That is due to the fact that these attributes are mostly related with
process and project kind of entities, for which there is not a well defined basic attribute.

Once the “What to Measure?” question was analyzed. The next step was to tackle the question: “How to measure?” To answer this question we gathered how the measurements of attributes in the selected papers were made and classified them in terms of the following characteristics: Representation, Description, Base or Derived Measurement, Scale (Fenton y Pfleeger, 1997), Empirically (Wohlin et al., 2000; Juristo and Moreno, 2001; Basili et al., 1999; Perry et al., 2000) or Theoretically (Weyuker, 1988; Briand et al., 1996; Whitmire, 1997; Zuse, 1998; Poels y Dedene, 2000) validated. This analysis is summarized in Table 5.

Let us have a look at the last characteristic, which has as its goal to discover if a measure has been validated empirically and/or theoretically. The aim of theoretical validation is to check whether the intuitive idea of the attribute being measured is considered in the defined measure. The main goal of empirical validation is to obtain objective information concerning the usefulness of the proposed metrics. Theoretical validation by itself is not enough to guarantee the usefulness of the measure, because it may occur that a measure is valid from a theoretical point of view, but it has no practical relevance in relation to a specific problem. As a consequence, a measure which has not been validated is not demonstrated to be useful. We therefore classified the measures in such a way as to know how many had been empirically and/or theoretically validated. This is shown in Figure 3.

As can be observed in Figure 3, about half of the measures found in the selected papers had been only empirically validated. This leads us to the conclusion that there is a great tendency to empirical validation. Furthermore, we can see that (24%) of the measurements had been validated only theoretically, although it was recognized in the papers that they need empirical validation. Finally only (20%) of the measurements had been both empirically and theoretically validated. It should be pointed out that it is necessary to get a common agreement to validated measures theoretically. Moreover empirically validation needs more data extracted from “real projects” in order to get practical conclusions.

Regarding the measurement focus found in the articles analysed, we have discovered the following approaches: Structured (Briand et al., 1996a), measurement focussing in Process, Object Oriented (OO) (Chidamber y Kemerer., 1994; Brito e Abreu y
Table 5: Definition of measure attributes.

<table>
<thead>
<tr>
<th>HOW?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation</td>
</tr>
<tr>
<td>Empirically/Theoretically, 103: 28%</td>
</tr>
</tbody>
</table>

Figure 3: Validated measures.

Carapuca, 1994; Lorenz y Kidd, 1994; Marchesi, 1998; Bansiya et al., 1999, 2002; Quality (Piattini y García, 2003), Function Points (IFPUG Release 4.2, 2004), UML (Marchesi, 1998), Complexity (McCabe, 1976; Henry y Kafura, 1981), Project (Putnam y Myers, 1992) and OCL (Reynoso et al., 2004). Figure 4 shows the amount of measurement in each approach. It shows us that the most supported approaches by measure are Object Oriented (OO) ones. This is due to this kind of projects are currently the most popular in software development. Continue with this part of the analysis, there are efforts to get a universal WEB measures definition, with this review we found conceptual models and frameworks in order to classify WEB measures.

Figure 4: Measure focus.

Finally, we analyzed the third question: When to measure?. To classify in what parts of the lifecycle project the measure must be taken for projects and process entities, the PMBOK guide (ANSI/PMI, 2004) was selected. In order to group when the measurements are taken for the product entity, the waterfall lifecycle model was applied. We chose these two models due to their wide acceptance and genericity. Figure 5 shows the proportion of product measurements in the different phases of the software life cycle:

Figure 5: Measure in life cycle software process.

As we can see in Figure 5, most measurements are carried out during the Design, Testing and Development phases of the waterfall lifecycle software process. In the Design phase, products such as architecture, system designs, requirements analysis, etc. are generated. Hence it is necessary to support this phase with measurements, in order to know characteristics of these products when carrying out the design. Moreover, measurement in the Design phase can support the future products to be generated, which mean that this phase is one of the most measured. Continue with this analysis, it should be pointed out that the Development phase is one of the most measured, because most of the software products are created here, such as: manuals, source code and, among other products, the software itself. Therefore, it is possible to collect quantity information about these products here. According to PSP (Humphrey, 2005), measures about size, effort, time, faults, defects, LOC, etc. are commonly taken in this phase. Another factor to take into account is that once the software system is created, it is necessary to validate if this system fulfils the quality requirements. The counting faults and deriving the reliability is the most widely applied and accepted method used to validate systems; most of this information focuses on the product and is commonly reported in terms of measurements. This is done in not only in the early phases but also especially in the testing phase, which is another of the most-measured phases in lifecycle software process.

In addition, the PMBOK guide defines the following general phases for project life: Initial, intermediate, and final phases. In Figure 6 we show the distributions of measures through these phases.
It is worth mentioning here that in the initial phase there could be sub-phases with one or more deliverables, according to the kind of project. In these sub-phases the following are usually measured: size, complexity, level of risk, cash, etc. Most measurements concentrate on the Initial phase, as in this phase the planning for the whole project is executed- this in turn constitutes the main effort in project management. In the Intermediate phase, many control activities are carried out in order to ensure the success of the project. Periodical reports are thereby generated with quantity information about process and project measures and indicators. For these reasons this phase is also one of the most measured in project lifecycle for project and process entities.

5 CONCLUSIONS AND FURTHER WORK

Software measurements are very important in software development process, because they help us, to control, estimate and improve process, projects and products, among other things. With that in mind, this article attempts to provide the state of art in software measurement, by carrying out a systematic review whose purpose is to summarize the most current and useful measures in the literature.

With this systematic review, we find out the following results:

(1) Measures are strongly aligned to product entity. Since this kind of entity has better attribute definition than project and product entities have, there are large amount of measures for the product. This leads to the conclusion that if an entity has a few measures, it is due to the fact that it doesn’t have specific attribute.

(2) Complexity gathered a great amount of measures because this attribute is used in different contexts. While size is also one of the most measured attributes since it is used in cost and development schedule estimation.

(3) There is a great tendency to obtain empirical validation. But it is necessary to get more data extracted from “real projects”, in order to get practical conclusions and to improve software quality.

(4) Development and Design are the most measured phases in lifecycle software process because it is in these phases that most software products are generated. It should be also noted that the testing phase is also one of the most measured phases. This is thanks to the fact that this phase involves quality activities for evaluating software quality characteristics, generally reported in terms of quantity values. But quality measures are considering in the early software development phases by counting faults which is the most widely applied method to determine software quality.

(5) For projects and process entities most measurements are concentrated in the Initial and Intermediate phases. That is because it is here that the project planning and control activities are developed.

(6) There are a large number of measures for OO projects. This is because these kinds of projects are currently the most popular in software development. Hence a lot of research has been done in this field.

(7) So many efforts had been made to get a universal WEB measures definition. In this review we found conceptual models and frameworks in order to classify WEB measures.

Finally, we need to relate the measurements found in this article to a specific software development process. The aim of this is to settle when a measure must be taken. To reach this goal, in our specific research, further work will take in the Process Model for the Software Industry (MoProSoft), which focuses on small companies and which is also the Mexican norm.

ACKNOWLEDGEMENTS

This article was supported by the Process Improvement for Promoting Iberoamerican Software the Competitiveness of Small and Medium Enterprises (COMPETISOFT) and Science and Technology for Development (CYTED).
REFERENCES


Towards Anchoring Software Measures on Elements of the Process Model

Bernhard Daubner and Bernhard Westfechtel
Bayreuth University, Applied Computer Science I
95440 Bayreuth, Germany
bernhard.daubner@uni-bayreuth.de
bernhard.westfechtel@uni-bayreuth.de

Andreas Henrich
Bamberg University, Media Informatics
96045 Bamberg, Germany
andreas.henrich@wiai.uni-bamberg.de

Keywords: Automatic software measurement, anchoring software measures, process models, measurement tool, maven.

Abstract: It is widely accepted that software measurement should be automated by proper tool support whenever possible and reasonable. While many tools exist that support automated measurement, most of them lack the possibility to reuse defined metrics and to conduct the measurement in a standardized way. This article presents an approach to anchor software measures on elements of the process model. This makes it possible to define the relevant software measures independently of a concrete project. At project runtime the work breakdown structure is used to establish a link between the measurement anchor points within the process model and the project entities that actually have to be measured.

Utilizing the project management tool Maven, a framework has been developed that allows to automate the measurement process.

1 INTRODUCTION

Effective management of software development projects requires permanent assessment of both the actual projects and the underlying development processes. Managers need to control projects quantitatively in order to maximize estimation accuracy based on historical project data, to minimize risks and time-to-market, and to reliably reproduce the related processes (Auer et al., 2003). As the continuous collection and analysis of measurement data is crucial in tracking and managing software development processes efficiently, the measurement process must be automated by proper tool support whenever it is possible and reasonable.

Today many tools are available to compute certain measures about software artifacts like source code, to store measurement data in a database and to create the corresponding reports. Also general frameworks and guidelines have been proposed to integrate software measurement into the software engineering environment (Basili and Rombach, 1988), (Kempkens et al., 2000), (Münch and Heidrich, 2004). Finally there exist measurement approaches like GQM (Basili et al., 1994) and PSM (McGarry et al., 2002) that support the project manager to define what measures to collect.

Within the above mentioned measurement frameworks it is often stressed that models and metrics defined for the whole organization should be reused to make measurement results comparable and to build up an experience base that can be used to assess future projects. In order to assure comparability of measurement results, measurement must be integrated into the software development process for a new project before project execution starts (Kempkens et al., 2000).

Concerning these last two aspects we think that there exists no proper tool support that provides both automated software measurement and reuse of already defined metrics. Moreover, it should be possible to implement the measurement program on top of an existing software development process. Hence, we suggest to anchor the software measures on elements of the applied process model. Thus the software measures that have to be collected can be determined ex ante and independently of a concrete project. This allows to define in advance, which software measures have to be computed on every software development project in order to make these projects comparable.

We will show within the next section that this approach is fairly new since the other published approaches either lack tool support for automated mea-
measure or are based on a special process model that is needed in order to provide automatic measurement.

2 RELATED WORK

The need for software measurement has been widely accepted and there exist many commercial software measurement tools. In (Auer et al., 2003) a survey on up-to-date measurement tools has been conducted. Only two out of five products were able to collect measurement data automatically. These tools (MetricCenter\(^1\) and ProjectConsole\(^2\)) mainly focus on software measures concerning the whole project (number of requirements, number of defects, etc.). These kinds of tools are certainly useful in order to provide a management view on ongoing projects and are also described in terms of project dashboards (Selby, 2005). But we think that they hardly provide support for reuse of the defined metrics to enable more fine granular comparisons between projects. This aspect is addressed by the Software Project Control Centers described in (Münch and Heidrich, 2004) including a reference model of concepts and definitions around SPCCs. A tool support for the proposed architecture however is not yet available.

In (Lott, 1996) a framework is presented to integrate measurement and process models in a way that supports automation of “measurement-based feedback”. Automated support for measurement-based feedback means that software developers and maintainers are provided with on-line and detailed information about their work. Lott has realized his approach by implementing a dedicated process-centered software engineering environment. Hereby the system informs the users about the tasks that they are expected to perform and collects data that are associated with the realization of these tasks. However most of the data is not measured automatically but must be provided by the user by means of a form-based interactive tool.

Also the APEL (Abstract Process Engine Language) (Dami et al., 1998) has been developed as process modeling and controlling framework with integrated measurement. It allows the user to link the measurement model with the process model in order to control which measures should be collected at what stages in the process. Since APEL acts as a workflow engine the automatic support it provides only works if the processes are exactly executed in the way they are specified.

In (Kempkens et al., 2000) a framework for integrated tool support within measurement programs is presented, which gives guidelines for setting up measurement tool support for software development processes. Hereby the framework allows companies to use their existing tools and processes. Within the presented case studies however much of the data collection was done manually and no evidence has been given for the demanded “reuse of models and metrics”.

Finally, a contrary approach has been presented by Johnson within his conference paper “You can’t even ask them to push a button: […]” (Johnson, 2001), where he introduced the approach of his tool Hackystat. The Hackystat software uses sensors to gather the activities of each user within the software development environment. Thus it analyses and logs what kind of work (programming, testing, modeling, etc.) the user is just performing. Hereby the system acts unobtrusively and the user is in no way interrupted within his work. It is however not easy to interpret the vast amount of data collected by Hackystat in a meaningful way.

3 ANCHORING SOFTWARE MEASURES

3.1 Software Measures in the Context of Process Models

Our aim is to describe a standardized way to integrate software measurement into the software development process in order to make software development projects comparable with respect to the applied software measures and the way the software measures are collected.

The crucial point is that especially those software measures that are relevant to the project at management level are fixed independently of the concrete project. Software measures that generally have to be collected are for instance determined at the introduction of a company wide measurement campaign (McGarry et al., 2002), are derived from tactical or strategic company goals with frameworks like the Goal Question Metric Paradigm (Basili et al., 1994) or are determined by a process evaluation methodology like the Capability Maturity Model Integration (CMMI) (Chrissis et al., 2003).

This insight leads directly to our approach to anchor the software measures on elements of the process model, because the process model is the basis that all the projects we want to measure will build on.

The process model structures the software development process into units typically called phases, disciplines or activities. Here we can distinguish between fine granular process models like the V-Modell...
XT\(^3\), which serves as the standard process model in Germany for managing government IT development projects, and more "pragmatic" process models like the Rational Unified Process (Kruchten, 2003) that contain guidelines about the workflows to operate on. One interesting question in this regard is how certain work efforts are distributed over the activities. For example one would expect that most of the programming effort is spent on activities belonging to the implementation phase. But is is not unlikely that programming efforts also are carried out within the scope of analysis related activities if requirements have to be clarified by means of prototypes. It might even be possible, if the development is accomplished in an agile way, that most of the programming effort is accumulated during analysis activities.

In order to monitor this we suggest to anchor the appropriate size measures on the relevant elements of the process model. Thus, we get software measures like

- **LOC** within the scope of programming related activities and
- **LOC** within the scope of analysis related activities

These measures use elements of the process model as anchor points and not concrete artifacts.

### 3.2 The Work Breakdown Structure

This yields to the demand to identify the artifacts that have been created or modified within the scope of a certain activity of the process model in order to measure their sizes respectively. For this purpose we utilize the work breakdown structure of the project as link between the process model which the project is based on and the above mentioned artifacts.

The work breakdown structure (WBS) (Project Management Institute, 2001) represents the structure of a project and contains the essential relationships between the elements of a project. The project is structured in a hierarchical manner into subtasks and work packages. This leads to a tree structure as shown in Figure 1, which contains all project activities that have to be performed within the individual development phases.

#### Identifying Artifacts Utilizing WBS Codes

Since the WBS is based upon the activities of the process model, we can assume that for each activity of the process model a corresponding work package exists within the work breakdown structure. Thus we can reduce the above demand to identify the artifacts associated with a certain activity to the determination of the artifacts that have been created within a certain work package. This however is possible by means of the identifying key (**WBS Code**) that usually each element of the work breakdown structure is associated with (Project Management Institute, 2001). Using for instance a *decade code scheme* (Figure 1) the number of digits unequal to 0 within the WBS code determines the position of the WBS element within the hierarchical work breakdown structure.

The WBS codes not only identify the work packages within the work breakdown structure. They also can be used to label the artifacts that have been created or modified within a particular work package. For that purpose it is necessary to maintain all artifacts under version control. Whenever a *commit* statement for a modified version of an artifact is executed within the software configuration management (SCM) system, the WBS code of this work package has to be stored together with the change-log information.

Thus by means of the WBS code within the change-log information of the SCM system the artifacts and also the changes of the artifacts are associated with the corresponding activities of the process model. And software measures that have been anchored on elements of the process model can be applied respectively.

#### Cross-Project Software Measurement

Since companies usually perform several similar software development projects, in the course of the time a standard work breakdown structure is established based on the company-specific process model. The standard WBS serves as basis for the concrete work breakdown structures used in the individual projects (Futrell et al., 2002). For the latter this standard work breakdown structure has to be extended or shortened respectively.

---

3\(\)http://www.v-modell-xt.de
The common elements of the WBS however keep hold of the same WBS code. Utilizing this standard work breakdown structure makes the application of software measures anchored on the process model also in a cross-project manner possible. Those software measures that are associated with the company-specific process model can be collected by means of the cross-project uniform WBS codes of the corresponding subtasks and work packages.

3.3 Agile Software Development

In Section 3.1 we have explained our approach considering well defined process models with a clear and elaborated structure. This immediately leads to the question whether this approach also is applicable on less elaborated or less process based development methodologies. Particularly within agile development methodologies it is not very common to establish a work breakdown structure (Beck, 2004), (Cockburn, 2001) that our approach needs to identify the artifacts.

The apparently obvious idea to collect software measures within the scope of the individual iterations of the agile development process does not lead to any benefit in order to compare several development projects or to assess the progress of the ongoing project.

But also software projects using an agile development style can be structured into at least two dimensions. First of all there are several kinds of activities to distinguish that are performed during a development project. Ambler for example mentions in the style of the RUP the “agile disciplines” Modeling, Implementation, Test, Deployment, Configuration Management, Project Management and Environment (Ambler, 2002). In addition to that the software units that are created can be categorized according to their function. Here one can think of application logic, presentation, data management, transaction management, logging etc. This means that the activities of an agile software project can be mapped into a matrix with the axes discipline and function. Therefore also the artifacts produced within an agile project can be tagged within the SCM system with 2-tupels that inform about the discipline they have been created in and the function they have.

Thus, regardless of the existence of a work breakdown structure, the artifacts under version control and the time records within the time recording system can be tagged with codes that identify the implemented functionality and the discipline within that scope the effort has been performed. These codes provide a “generic project plan” and we can use the elements of this “generic WBS” to anchor our software measures on. In Section 4.3 we will demonstrate this agile application of our measurement approach on a small development project.

3.4 Preferred Usage of Our Approach

We think that this measurement approach is preferably suitable for measures that are relevant for project controlling. Here often the measures size, time, effort, defects and productivity are mentioned (Russac, 2002), (Putnam and Myers, 2003). For these measures our approach supports automatic measurement and reuse.

The only thing our approach needs is that all projects are based upon a consistent skeletal structure. This can be a standard WBS for process-based development projects or a discipline-function-matrix for agile projects. The skeletal structure provides the numbering scheme in order to tag the produced artifacts within the SCM system or administrative information like time records within the time recording system.

4 INTEGRATING AND AUTOMATING SOFTWARE MEASUREMENT

4.1 The Project Management Tool Maven

In order to continuously measure the software development process and to get the required feedback on process improvement attempts the collection of the software measures has to be automated and must be integrated into the development process. Implementing our approach we have restricted ourselves to Java projects. This allows us to use the software Maven from the Apache Project4. Maven is a tool that supports the management and the development process of Java projects. On the one hand it supports the developer at the so called build process, i.e. Maven compiles the source code and considers thereby dependencies like additional libraries that have to be made available first. With this respect Maven can be regarded as an alternative to Ant5.

In addition to that it provides means to hold important information about project members, project resources (e.g. documents, licences) and about the software configuration and bug tracking systems that are used within the project. Using plugins, the functionality of Maven can be extended anytime. The plugins thereby can access the project information and thus for instance access the software configuration

4http://maven.apache.org
5http://ant.apache.org
system in a generic manner without the need to know the exact type of the SCM software.

4.2 Measurement Transmitters for Software Measures

At this point we join and use Maven plugins in order to implement what we call measurement transmitters. In climate research measurement transmitters are used for instance to measure rainfall. A software measurement transmitter however is a Maven plugin that is capable of computing a certain software measure. For example a JavaLOC measurement transmitter is able to determine the Lines of Code of Java program files. And as a rainfall measurement transmitter can be set up at different locations the JavaLOC measurement transmitter can likewise be configured in that way with the Maven XML configuration file that only such code lines are counted that have been created within the scope of a certain element of the process model.

We do this by anchoring the measurement transmitter on the corresponding work package of the WBS using the XML configuration file of the plugin.

Querying the software configuration management system, which also is referenced within the Maven configuration, and utilizing the change-log information, it is possible to identify such source code files (including the revision numbers) that have been created or modified within the scope of the mentioned work packages. Now the affected versions of these source files successively have to be checked out in order to measure their changes in size in LOC.

Measurement transmitters also can be combined. For example a JavaLOC measurement transmitter can be combined with an Effort measurement transmitter which leads to a simple Productivity measurement transmitter. First the JavaLOC measurement transmitter has to determine the code size. Then the Effort measurement transmitter determines the expenditure of time by querying the time recording database. The results refer to the same work packages. Thus, the productivity during the treatment of these work packages can be computed.

4.3 The Maven Measurement Framework

We call our implementation of this measurement approach on the top of Maven the Maven Measurement Framework (MMF). It is a framework because it consists of a collection of API functions that are invoked by Maven plugins which thereby implement the measurement transmitters. The API provides the application logic like analysing the change-logs of the SCM system, querying the time recording database for effort information about certain work packages or applying software measures on source code files. Thus we can provide Maven plugins for arbitrary software measures by combining these API calls respectively. Within the Project Object Model, the central XML configuration file of all Maven projects, the project manager can select which Maven plugin of our framework should be used and what parameters should be passed to it, in order to define the measurement scope.

Example Use of the Maven Measurement Framework

We have evaluated our approach on two software development projects that have been conducted by students within the bachelor’s study courses in computer science at Bayreuth University. Two student teams had to implement a client for a Mancala game\(^6\). As the corresponding server containing the data for the actual state of the game was alread on-hand the students essentially had to implement the GUI and the application logic for the client. The latter also comprised the implementation of a game tree in order to find possible moves.

With our measurement framework we have monitored the development progress of the two teams (Figure 2). The size of the Java sources is counted in Non Commenting Source Statements\(^7\). It can be seen that team 1 at first has put its focus on the development of the GUI and has implemented the application logic not until the last four days. On the contrary, team 2 has equally developed both components in parallel from the beginning. This measurement based analysis has been verified by the resulting programs. It could be seen that the GUI of team 1 was more elaborate, whereas the program of team 2 was technically superior.

![Development Progress](http://www.kclee.de/clemens/java/javancss/)

\(^6\)http://en.wikipedia.org/wiki/Mancala
\(^7\)http://www.kclee.de/clemens/java/javancss/
5 CONCLUSION

The automatic collection, interpretation and visualization of software measures is a complex task. The aim of this approach is to provide a lightweight tool for project managers to assess their development projects and the applied software process. The anchoring of software measures on elements of the process model enables the project manager to define software measures in advance before a concrete project starts. This allows for comparison of distribution ratios of effort and time spent on the activities of several projects in order to assess the progress of the current project or the productivity of the individual projects.

Therefore we think that with our approach to anchor software measures on elements of the process model and to identify by means of the work breakdown structure the entities that have to be measured we have found a practicable tradeoff between the automatic collection of information and the additional effort the developers have to perform.

The developers have to record their activities by indicating the associated WBS code within the time recording system. Furthermore these references to the work breakdown structure must also be maintained within the software configuration management and the bug tracking system. This is however a common modus operandi (Selby, 2005).

With our approach we gain the possibility to collect certain software measures in a standardized way that allows the cross-project comparison of the measurement results, because we define the entities to measure already before the start of the individual projects on the basis of the process model. The actual computation of the software measures is realized automatically at project runtime using Maven.

REFERENCES


WEB METRICS SELECTION THROUGH A PRACTITIONERS’ SURVEY

Julian Ruiz, Coral Calero, Mario Piattini
Alarcos Research Group, Information Systems and Technologies Department, UCLM-Soluziona Research and Development Institute, University of Castilla-La Mancha, Paseo de la Universidad, 4, 13071 - Ciudad Real, Spain
Julian.Ruiz, Coral.Calero, Mario.Piattini@uclm.es

Keywords: Web Metrics, Quality.

Abstract: There are a lot of web metrics proposals. However, most previous work does not include their practical application. The risk of doing so, is to limit all the effort made just to an academic exercise. In order to eliminate this gap as well as to be able to apply the work developed, it is necessary to involve the different stakeholders related to web technologies as an essential part of web metrics definition. So, it is crucial to know the perception they have about web metrics, especially those related to the development and maintenance of web sites and applications. In this paper, we present the work we have done to find out which web metrics are considered useful by web developers and maintainers. This study has been performed on the basis of the 385 web metrics classified in WQM, a Web Quality Model defined in a previous work, using as validation tool, a survey made by professionals of web technologies. As a result, we have found out that the most weighted metrics were related to usability. That means that web professionals give more importance to the user of metrics than to their own effort.

1 INTRODUCTION

The spectacular development of the web has led to an increasing importance of related technologies in the functioning of organizations as well as in people’s lives. It is compulsory that developed products, both complex applications or simple web sites, satisfy a minimum quality standard (Cutter Consortium, 2000).

In the field of web metrics, a large research effort has been made with proposals from very diverse perspectives.

In spite of the fact that some of the proposed metrics have not been formally defined or theoretically or empirically validated, fortunately, in the last years, the tendency is changing and justification, formalization and validation are also taken into account (Abrahão et al., 2003).

However, most of the work is academic and doesn’t take into account industrial concerns. There are some exceptions. Among them, we can cite the works (Reifer, 2000, 2002) and (Mendes et al., 2003, 2005).

To eliminate the gap between practical application and academic world, it is necessary to better involve the different actors related to web technologies. To do so, it is essential to know the perception of web metrics that these actors have.

With this objective, we have performed a survey among web technologies professionals to select the metrics that are considered interesting or useful by them. Once we have the set of metrics, we could measure the web sites or the web applications and obtain corresponding quality indicators.

Our starting point is to consider the metrics that we collected from the literature and were used in our previous study (Calero et al., 2005) in which we classified a total of 385 web metrics, using the WQM quality model.

In the next section, we will expose the criteria followed in the metrics selection that we have used in our survey. In the third section, we will present the survey, its results and conclusions. Finally, in the fourth section, future work will be stated.

2 INTERNAL METRIC SELECTION

As it is not possible to prepare a survey including the 385 metrics classified in WQM, we performed a
first selection with the objective of restricting them into a manageable and representative set of web metrics to be included in our survey.

2.1 Selection Criteria

The selection was made taking into account the following considerations:

a) The number of metrics must be as limited as possible.
b) The selection must cover the different perspectives to be considered.
c) The most relevant works must be examined in a detailed way, especially those having an experimental component.
d) Works based on a concrete methodology must not be refused at the beginning but we will have to bear in mind the possibility of an easy generalization.
e) Given that many aspects such as usability have a great number of metrics, we will have to make an even bigger synthesis effort than with other aspects in which it is clear the lack of metrics.
f) It is not necessary that the selected number of metrics must be proportional to the number of collected metrics per aspect.

According to these considerations, we will establish the following criteria for the selection process:

1. To select those metrics that are proposed in several works.
2. To select those metrics that represent simple concepts.
3. To avoid duplicities, eliminating as much as possible metrics that could be assimilated into others, with respect to meaning, even not representing the same concept.
4. To eliminate metrics coming from the specialization of other metrics. Although they can allow us to measure certain characteristics in a more precise way, they can also made us lose a more general vision.
5. To incorporate some metrics that are not very common with the purpose of introducing variability.
6. To incorporate metrics specific for a methodology but able to be adapted to others.

Furthermore, if at any time there is a contradiction between the criteria, we will prioritize the simplest one.

2.2 Metrics Selection

As we have already indicated, our starting point has been the 385 metrics.

According to criterion 1, we have a set of metrics proposed by a large number of authors and that have also a very simple meaning (criterion 2) such as Number of Web Pages, Depth, Breadth, Number IN Links, Total Number of Links, Number of Broken Links, %Broken Links, Total Number of Images, Images per Page, Images with ALT Text, all of them with respect to the Website or Web Application, and Download Time (of a page), Links of a Page and Images of a Page.

We have also included others such as Compactness, Stratum, and Cyclomatic Complexity, based on criterion 5.

Following criteria 1, 2 and 5, we have included the following: Quick Access Pages, Site Map, Global Help, Scoped Search, Stability, Link Colour Style Uniformity, Global Style Uniformity, Foreign Language Support, Contact Address. And we have selected other generic metrics (criterion 2) like Suitable Information and Updated Information.

Concerning usability, as most of metrics are the result of a specialization (criterion 4), we have extracted the following (remember that other metrics have been already included from other works): Display Colour Count, Text Positioning Count, Text Cluster Count, Font Count and Reading Complexity, all of them with respect to a web page.

Regarding works related to the development and maintenance of web applications, we have selected (we have not included those included above), following criterion 2: Media Count, Program Count, Total Page Allocation, Total Media Allocation, Total Code Length, Page Allocation, Media Duration, Media Allocation, Code Length (LOC), Code Comment Length, Reused Media Count, Reused Program Count, Total Reused Media Allocation, Total Reused Code Length, Reused Code Length, Reused Comment Length, Total Page Complexity, Page Complexity, Audio Complexity, Video Complexity, Animation Complexity, Scanned Image Complexity, Total Effort (Design&Auth), Total Page Effort, Total Media Effort, Program Effort, Experience, and Tool Type. And others like Total Number Flash Animations, Total Number of Icons/Buttons, Average Length Audio Clips, Average Length Video Clips, Reused Web Pages, and Reused Docs.

We include, according to criterion 6, (all of them with respect to the web application): Web Building Blocks, Number of COTS Components, Number of
Object or Application Points, Number of XML, SGML, HTML and Query Language Lines, Number of Web Components, Number of Scripts (Visual Language, Audio, Motion) and Number of Web Objects. And based on criteria 5, we have taken the metric Peak Staff.

Besides, due to the application of criterion 4, we take into account the model efforts and the total effort: Total Design Effort, Information Effort, Navigation Effort and Presentation Effort.

Following criterion 2, we select Server Scripts, Client Scripts, Web Page Scripts, Web Page WebObjects, Total Languages and Page Languages.

We have not considered other metrics that are not relevant as compared to the selected ones because they mean an excessive specialization (criterion 4).

With this selection we have obtained 85 metrics, to be included in the survey (see appendix).

3 SURVEY

In this section, we will deal with aspects related to the survey, its objective, design, obtained results and its discussion.

3.1 Definition of Objectives

We have focused on the following objectives:

- To determine the importance given by web professionals to the considered web metrics.
- To study the impact of participant experience on the importance of metrics.
- To identify other aspects not taken into account in our work and that are considered important by web practitioners.
- To identify the concordances/discordances with the metrics proposed by researchers in the literature.

3.2 Survey Participants

An important aspect to be considered is who the survey target since any community has its own characteristics. For our purposes, our survey is addressed to practitioners involved in tasks of developing or maintaining applications and web systems with diverse degree of experience.

Thus, the technical concepts should not represent any problem. However, to fulfill our purpose, we have to take into account other aspects. For example, if the survey comes from the academic field can be seen by web technologies professionals as it does not fulfill their needs and they can refuse to fill it out. Subjects were not involved only in a passive way. In addition to theirs answers we tried to involve them in the project by soliciting suggestions from them.

For this survey our objective population is the web professionals. For the sample, we have considered professionals that previously we had maintained some contact in the past (or with their companies). Choosing them to conserve the diversity in the applications developed (scope of work), its experience degree, and the companies for whom work.

3.3 Survey Design

To fulfill the fixed objectives, we have structured our survey into three parts:

A. Data of the Subject.
B. Web Metrics.
C. Suggestions.

We have to take into consideration that the survey design is conditioned by the high number of metrics to be included in it. Answering a survey with questions about 85 metrics carries out certain reticences regarding the necessary time to fill it out.

Now, we will deal with each part separately.

3.3.1 PART A: Data of the Subject

There are a great variety of web professionals depending on their experience, their job and the technologies that they use. For these reasons, and following the recommendations of (Pfleeger and Kitchenham, 2001) and (Kitchenham and Pfleeger, 2002a-d, 2003) we have included generic questions about personal data. Thus, part A, Data of the Subject, is composed of three questions:

1) Job (Developer, Maintenance Manager, Others)
2) Years of Experience
3) Category of the developed product:
   a) Web site with static pages
   b) Web sites with dynamic generation of pages, working with jsp, php, asp, within a centralized environment (e.g. applications for small or medium size enterprise)
   c) Web sites using Content Management Systems (such as CMS of Microsoft, Zope, Tipo3,…) in a distributed environment (e.g. applications for a corporation)
3.3.2 PART B: Web Metrics

To make part B as simple as possible, we have decided to use close questions, one for each metric, quantifying the importance of each metric by using a Likert scale with an interval from 0 (not important) to 9 (very important).

To avoid fatigue and motivation loss we decided to start the questioner by simple metrics.

For providing a common background to the subjects, we included in the documentation a minitutorial about the metrics of the survey.

3.3.3 PART C: Suggestions

This part has two open questions. The first is to include suggestions of other metrics that subjects consider interesting, and the second is to include suggestions about the survey. The objective of this part is, on the one hand, to detect metrics that we have not considered, and on the other hand, to validate our survey.

3.4 Results

The survey was sent to the subjects by personalized email, avoiding as much as possible to give the impression of being like a circular to avoid the rejection rate. The survey could be sent back once filled out in electronic or paper format.

66 surveys were sent and we obtained 42 answers (63.6%), during the ten days deadline.

Two participants classified themselves as web users and the rest as professionals with different degrees of experience and different development environment. Then, the sample of web technologies is formed by 40 subjects. From them, we have centred our study as follows.

In the tables 1-4, we show the obtained results for the different subjects categories. In each one of the tables, column Score shows the arithmetic average given by subjects of each considered group, and Deviat. (or Dev) its standard deviation. As we have noted above, each metric has been scored into a scale from 0 (not important) to 9 (very important). The interpretation of the results shown here will be analysed in the following section.

For the set of the 40 subjects, the most accepted metrics are shown in table 1. For those related to technologies of static web pages (Category a, with only 6 subjects), we obtain table 2 (we have included this table in spite of the reduced sample size). Regarding the most valued metrics for categories b and c of subjects (34 subjects), we obtain table 3.

In the table of the appendix, we can see the complete relation of all the metrics studied in the survey ordered according the importance given by these 34 subjects of group b-c. Column Score is the average score by the metrics, and Dev. its standard deviation, for each of the group considered.

Table 1: Metrics rank for all the subjects (40 subjects).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Score</th>
<th>Deviat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated Information</td>
<td>8.35</td>
<td>1.05</td>
</tr>
<tr>
<td>Suitable Information</td>
<td>8.13</td>
<td>1.26</td>
</tr>
<tr>
<td>Download Time</td>
<td>8.05</td>
<td>1.11</td>
</tr>
<tr>
<td>Global Style Uniformity</td>
<td>7.88</td>
<td>1.11</td>
</tr>
<tr>
<td>Scoped Search</td>
<td>7.65</td>
<td>1.69</td>
</tr>
<tr>
<td>Link Colour Style Uniformity</td>
<td>7.58</td>
<td>1.32</td>
</tr>
<tr>
<td>Navigation Effort</td>
<td>7.55</td>
<td>1.50</td>
</tr>
<tr>
<td>Information Effort</td>
<td>7.48</td>
<td>1.55</td>
</tr>
<tr>
<td>Total Effort (Design)</td>
<td>7.40</td>
<td>1.58</td>
</tr>
<tr>
<td>Presentation Effort</td>
<td>7.38</td>
<td>1.56</td>
</tr>
<tr>
<td>Developer’s Experience</td>
<td>7.38</td>
<td>1.58</td>
</tr>
<tr>
<td>Quick Access Pages</td>
<td>7.38</td>
<td>1.50</td>
</tr>
</tbody>
</table>

Table 2: Metric rank for static web page developers.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Score</th>
<th>Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suitable Information</td>
<td>8.33</td>
<td>0.82</td>
</tr>
<tr>
<td>Updated Information</td>
<td>8.33</td>
<td>0.82</td>
</tr>
<tr>
<td>Foreign Language Support</td>
<td>8.00</td>
<td>0.89</td>
</tr>
<tr>
<td>% Broken Links</td>
<td>7.83</td>
<td>1.60</td>
</tr>
<tr>
<td>Global Style Uniformity</td>
<td>7.67</td>
<td>1.03</td>
</tr>
<tr>
<td>Link Colour Style Uniformity</td>
<td>7.50</td>
<td>0.84</td>
</tr>
<tr>
<td>Number of IN Links</td>
<td>7.17</td>
<td>0.98</td>
</tr>
<tr>
<td>Number of Broken Links</td>
<td>7.17</td>
<td>1.83</td>
</tr>
<tr>
<td>Download Time</td>
<td>7.17</td>
<td>1.47</td>
</tr>
<tr>
<td>Global Help</td>
<td>7.00</td>
<td>1.10</td>
</tr>
<tr>
<td>Contact Address (e-mail, phone, mail)</td>
<td>7.00</td>
<td>0.63</td>
</tr>
<tr>
<td>Scoped Search</td>
<td>7.00</td>
<td>1.10</td>
</tr>
</tbody>
</table>

Table 3: Metric rank for b-c subjects category.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Score</th>
<th>Deviat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated Information</td>
<td>8.35</td>
<td>1.10</td>
</tr>
<tr>
<td>Download Time</td>
<td>8.21</td>
<td>0.98</td>
</tr>
<tr>
<td>Suitable Information</td>
<td>8.09</td>
<td>1.33</td>
</tr>
<tr>
<td>Global Style Uniformity</td>
<td>7.91</td>
<td>1.14</td>
</tr>
<tr>
<td>Scoped Search</td>
<td>7.76</td>
<td>1.76</td>
</tr>
<tr>
<td>Navigation Effort</td>
<td>7.74</td>
<td>1.26</td>
</tr>
<tr>
<td>Information Effort</td>
<td>7.68</td>
<td>1.36</td>
</tr>
<tr>
<td>Total Effort (Design)</td>
<td>7.62</td>
<td>1.30</td>
</tr>
<tr>
<td>Link Colour Style Uniformity</td>
<td>7.59</td>
<td>1.40</td>
</tr>
<tr>
<td>Presentation Effort</td>
<td>7.56</td>
<td>1.33</td>
</tr>
<tr>
<td>Developer’s Experience</td>
<td>7.53</td>
<td>1.54</td>
</tr>
</tbody>
</table>
We have divided the group b-c in two subgroups: the first one is formed by subjects with at least three years’ experience (19 subjects), and, the other one is formed by those subjects with less than three years’ experience (15 subjects). In the rest of the section we will refer to them as subgroups I and II), respectively.

We have decided to perform this division since we think that from three year’s experience there is a qualitative leap in developer maturity as well as in their knowledge of the technologies they work with.

For the sake of clarity, in table 4, we have extracted the results corresponding to the best scored metrics (score upon 7.50) according to subgroup I (three or more years experience). As we can see results do not differ very much with respect to those obtained by complete category b-c (see table 3).

Table 4: Metrics rank for practitioners with 3 or more years of experience in categories b-c.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Score</th>
<th>Deviat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Updated Information</td>
<td>8.37</td>
<td>1.12</td>
</tr>
<tr>
<td>Download Time</td>
<td>8.32</td>
<td>0.95</td>
</tr>
<tr>
<td>Suitable Information</td>
<td>8.21</td>
<td>1.18</td>
</tr>
<tr>
<td>Scoped Search</td>
<td>8.16</td>
<td>1.01</td>
</tr>
<tr>
<td>Total Effort (Design)</td>
<td>8.11</td>
<td>0.94</td>
</tr>
<tr>
<td>Quick Access Pages</td>
<td>8.05</td>
<td>0.97</td>
</tr>
<tr>
<td>Information Effort</td>
<td>7.95</td>
<td>1.13</td>
</tr>
<tr>
<td>Total Effort (Design&amp;Auth)</td>
<td>7.95</td>
<td>1.03</td>
</tr>
<tr>
<td>Global Style Uniformity</td>
<td>7.89</td>
<td>0.94</td>
</tr>
<tr>
<td>Navigation Effort</td>
<td>7.89</td>
<td>1.20</td>
</tr>
<tr>
<td>Presentation Effort</td>
<td>7.79</td>
<td>1.13</td>
</tr>
<tr>
<td>Number of Broken Links</td>
<td>7.79</td>
<td>1.96</td>
</tr>
<tr>
<td>Developer’s Experience</td>
<td>7.74</td>
<td>1.05</td>
</tr>
<tr>
<td>Page Allocation</td>
<td>7.53</td>
<td>2.06</td>
</tr>
<tr>
<td>Program Effort</td>
<td>7.53</td>
<td>1.47</td>
</tr>
<tr>
<td>Total Page Effort</td>
<td>7.53</td>
<td>1.43</td>
</tr>
</tbody>
</table>

Furthermore, if we compare the 22 most highlighted metrics by subgroups I and II, we obtain that there are more or less the same metrics except for only four metrics for each subgroup that they do not appear until positions 16th and 15th respectively (see appendix).

3.5 Discussion and Interpretation of Results

We notice that considerations about table 2 – category a), static web page developers– will be only indications, because of the reduced sample size.

The first conclusion we can extract is that Usability is very important. This result was foreseeable taking into account the importance of usability in Web Applications (Calero et al., 2005). Information quality is very important, Suitable Information and Updated Information. We also note the coincidence of groups a) and b-c) in other four metrics among the most valued also related to usability (Download Time, Scoped Search, Global Style Uniformity, Link Colour Style Uniformity).

The rest of the metrics in table 3 are related to effort (Information Effort, Navigation Effort, Total Effort, Presentation Effort and Developer’s Experience). It is paradoxical that developers prioritize the user vs. their effort.

In opposition to the general perception, our survey shows that importance granted in Literature to the number of pages and the number of images does not correspond to the perception that developers have of them.

Nevertheless, we do not mean that the number of pages is not important, since precisely the access to information is made from a page (or a data entry if it is carried out by other system). But these access pages would be the important ones and not those dynamically generated because the most important aspects are the programs that generate pages, the information they contain, how information is presented, and not the number of such pages that can be generated.

A similar reasoning can be made for the number of images. In a small website (and normally static), possibly image processing would be manual. But in a system with thousands of items, they would be provided in a digitalized format, probably in a database. Consequently, system complexity should be measured according to programs that use the database, not to the database size.

As we have already mentioned, to achieve the last cited objective, in section 3.1, we have incorporated into the survey, a third section of suggestions not only regarding metrics but also the survey itself.

With relation to the suggestion of metrics, we have found that almost all are also related to Usability and in particular, to Accessibility and adaptation to standards, compatibility with navigators and, in a lower degree, others related to performance and security.

3.6 Conclusions

In summary, the main conclusions we can extract from the survey are:

- Developers prioritize usability instead of their effort. By this, it is convenient to have tools that
from the first stages of development provide an estimation of product usability.

- Some metrics frequently used in Literature, have a relative importance for developers (e.g. number of pages and number of images). This is because the developed products are complex, with the use of dynamic generation of pages, and the use of Content Management Systems. There is a necessity of metrics and frameworks for that.

### 4 FUTURE WORK

As we said in the introduction, our work has the purpose of creating web applications quality indicators. We consider it essential to take into account the vision of quality of web professionals. To do so, we have divided our task into two stages, the first stage he consists in having a first approximation to metrics considered relevant by developers and sites and web applications administrators. As a result of this, we have obtained that the best valued metrics are Updated Information, Suitable Information, Download Time, Global Style Uniformity and Scoped Search.

The second stage consists of determining the importance of each metric with respect to each quality characteristic and each phase of the life cycle process. Therefore, we must obtain the metric weights with the purpose of obtaining quality indicators of a site or a web application.

However, before starting the second stage, we aim at carrying out the survey again using a different group of subjects to check the validity of the obtained results. The indicators obtained in our work must be able to be incorporated into web development methodologies.

Other aspect to be considered (considering the importance given to usability) is the incorporation of metrics for estimating the end product usability during the development. The same happens with accessibility.

### ACKNOWLEDGEMENTS

We would like to thank all participants in this survey for their time and above all, for their suggestions and especially to those that not only indicated suggestions in the survey but also came to talk to the authors about new approaches in our work. Thank you all.

This work is part of the CALIPO project (TIC 2003-07804-C05-03) and the CALIPSO network (TIN2005-24055-E) supported by the Spanish Ministerio de Educación y Ciencia and by the DIMENSIONS project (PBC-05-012-1) supported by FEDER and Junta de Comunidades de Castilla-La Mancha.

### REFERENCES


ICSOFT 2006 - INTERNATIONAL CONFERENCE ON SOFTWARE AND DATA TECHNOLOGIES

APPENDIX
Metric

Updated Information
Download Time
Suitable Information
Global Style Uniformity
Scoped Search
Navigation Effort
Information Effort
Total Effort (Design)
Link Colour Style Uniformity
Presentation Effort
Developer's Experience
Quick Access Pages
Page Allocation
Total Effort (Design&Auth)
Number of Broken Links
Global Help
Program Effort
Total Page Effort
% Broken Links
Foreign Language Support
Reading Complexity
Numb. of XML, SGML, HTML and Query Language Lines
Stratum
Total Media Effort
Site Map
Number of Web Page Scripts
Tool Type
Impurity Tree
Number of Reused Web Pages
Contact Address (e-mail, phone, mail)
Number of Server Scripts
Number of Client Scripts
Number of Web Objects
Total Reused Code Length
Reused Code Length
Total Code Length
Compactness
Reused Program Count
Number of Object or Application Points
Web Building Blocks
Number of Scripts (Visual Language, Audio, Motion)
Number of Web Components
Depth
Total Page Allocation
Code Length (LOC)
Total Number of Icons/Buttons
Number of COTS Components
Web Page WebObjects
Program Count
Display Colour Count
Breath
Peak Staff
Total Media Allocation
Code Comment Length
Text Positioning Count
Number of Web Pages
Media Allocation
Font Count
Text Cluster Count of a Page
Links of a Page
Number of Total Links
Number IN Links
Connectivity Density
Total Languages
Images with ALT Text
Images per Page
Total Number of Images
Media Duration
Number of Page Languages
Images of a Page
Reused Media Count
Cyclomatic Complexity
Average Length Video Clips
Total Reused Media Allocation
Total Number Flash Animations
Average Length Audio Clips
Animation Complexity
Reused Docs
Media Count
Page Complexity
Audio Complexity
Video Complexity
Total Page Complexity
Scanned Image Complexity
Reused Comment Length

244

Score in group b-c
Score in group b-c
Score in group b-c
Subgroup
Subgroup
Score in group a
(34 subjects)
Experience>=3 years Experience<3 years
(6 subjects)
(19 subjects)
(15 subject)
Aver.
Dev.
Aver.
Dev.
Aver.
Dev.
Aver.
Dev.
8.35
8.21
8.09
7.91
7.76
7.74
7.68
7.62
7.59
7.56
7.53
7.47
7.44
7.41
7.35
7.26
7.18
7.15
7.00
6.91
6.88
6.85
6.82
6.74
6.65
6.62
6.62
6.61
6.56
6.53
6.47
6.47
6.45
6.44
6.41
6.35
6.30
6.29
6.29
6.26
6.24
6.24
6.18
6.18
6.12
6.09
6.09
6.09
5.97
5.94
5.85
5.85
5.82
5.76
5.76
5.65
5.65
5.59
5.58
5.53
5.47
5.44
5.44
5.41
5.35
5.29
5.21
5.12
5.06
5.03
4.94
4.91
4.91
4.91
4.85
4.85
4.82
4.79
4.71
4.65
4.53
4.50
4.32
4.09
3.85

1.10
0.98
1.33
1.14
1.76
1.26
1.36
1.30
1.40
1.33
1.54
1.56
2.08
1.37
2.09
1.36
1.38
1.48
2.15
2.27
1.56
1.26
2.01
1.99
1.86
1.41
1.91
1.80
1.93
2.33
1.67
1.71
1.66
2.02
1.99
1.74
1.47
1.90
1.43
2.14
1.74
1.37
1.87
2.38
1.85
1.83
1.64
1.81
1.96
2.20
1.84
2.09
2.47
2.13
2.35
1.94
2.41
1.97
2.22
2.09
1.94
2.06
2.27
1.97
2.44
2.14
2.25
2.48
1.97
2.10
2.51
2.40
2.35
2.50
2.41
2.38
2.34
2.33
2.18
2.10
2.25
2.36
2.20
2.02
2.28

8.37
8.32
8.21
7.89
8.16
7.89
7.95
8.11
7.47
7.79
7.74
8.05
7.53
7.95
7.79
7.26
7.53
7.53
7.21
7.00
7.44
6.74
6.94
7.32
6.95
6.63
6.53
6.56
6.58
7.21
6.21
6.53
6.39
6.58
6.42
6.63
6.39
6.26
6.26
6.21
6.26
6.32
6.47
6.26
6.32
6.37
6.21
6.11
6.05
6.00
5.95
5.84
5.58
6.32
5.58
5.89
5.47
6.05
5.50
5.26
5.47
5.37
5.37
5.32
5.84
5.89
5.53
4.74
4.89
5.32
4.79
4.89
4.53
4.89
5.26
4.47
4.89
4.95
5.05
4.68
4.58
4.58
4.58
4.32
4.26

1.12
0.95
1.18
0.94
1.01
1.20
1.13
0.94
1.26
1.13
1.05
0.97
2.06
1.03
1.96
1.28
1.47
1.43
2.23
2.13
1.04
1.05
2.29
1.57
1.68
1.30
2.06
2.09
1.46
1.93
1.84
1.26
1.33
1.68
1.77
1.57
1.69
1.59
1.33
2.07
1.82
1.16
1.58
2.28
1.80
1.64
0.98
1.45
1.54
2.36
1.84
2.34
2.22
1.63
2.48
2.02
2.32
2.04
2.53
2.18
2.06
2.14
2.52
2.31
2.36
1.85
2.17
2.18
2.18
2.00
2.37
2.62
2.12
2.49
2.40
2.14
2.40
2.15
2.09
2.24
2.24
2.46
2.32
1.95
2.47

8.33
8.07
7.93
7.93
7.27
7.53
7.33
7.00
7.73
7.27
7.27
6.73
7.33
6.73
6.80
7.27
6.73
6.67
6.73
6.80
6.20
7.00
6.67
6.00
6.27
6.60
6.73
6.67
6.53
5.67
6.80
6.40
6.53
6.27
6.40
6.00
6.20
6.33
6.33
6.33
6.20
6.13
5.80
6.07
5.87
5.73
5.93
6.07
5.87
5.87
5.73
5.87
6.13
5.07
6.00
5.33
5.87
5.00
5.67
5.87
5.47
5.53
5.53
5.53
4.73
4.53
4.80
5.60
5.27
4.67
5.13
4.93
5.40
4.93
4.33
5.33
4.73
4.60
4.27
4.60
4.47
4.40
4.00
3.80
3.33

1,11
1,03
1,53
1,39
2,34
1,36
1,59
1,46
1,58
1,53
2,02
1,87
2,16
1,49
2,18
1,49
1,16
1,45
2,09
2,51
1,82
1,51
1,68
2,27
2,05
1,59
1,75
1,45
2,45
2,55
1,42
2,20
2,03
2,43
2,29
1,93
1,21
2,29
1,59
2,29
1,70
1,64
2,18
2,58
1,96
2,05
2,25
2,22
2,45
2,07
1,91
1,81
2,80
2,52
2,24
1,84
2,59
1,77
1,88
2,00
1,85
2,03
2,00
1,51
2,46
2,29
2,37
2,82
1,71
2,23
2,75
2,19
2,61
2,60
2,41
2,64
2,34
2,61
2,28
1,99
2,33
2,32
2,07
2,14
1,99

8,33
7,17
8,33
7,67
7,00
6,50
6,33
6,17
7,50
6,33
6,50
6,83
6,00
5,83
7,17
7,00
5,17
5,67
7,83
8,00
6,83
6,00
6,00
4,33
6,50
6,50
5,83
6,83
6,67
7,00
6,50
6,83
6,33
6,17
5,83
6,17
6,67
6,33
6,67
5,50
6,17
6,50
6,83
5,33
5,83
5,67
6,33
6,50
5,50
6,50
6,17
5,83
6,17
6,33
6,33
6,33
5,67
5,67
6,17
6,00
6,17
7,17
6,17
4,83
5,50
6,00
4,67
4,17
5,17
4,83
6,33
6,67
4,50
6,00
3,83
4,33
4,33
6,17
4,67
6,00
4,17
4,00
3,67
3,17
3,83

0,82
1,47
0,82
1,03
1,10
2,35
2,16
2,48
0,84
2,42
1,64
0,98
2,28
3,06
1,83
1,10
2,40
2,88
1,60
0,89
2,56
1,67
1,79
2,58
0,84
1,52
2,14
1,47
1,21
0,63
1,52
1,17
1,21
1,72
1,47
1,17
1,21
1,86
1,03
0,84
1,60
1,05
1,60
3,01
1,47
1,51
0,52
1,05
1,97
1,52
1,94
1,83
2,14
1,51
1,63
1,63
1,97
1,51
0,98
1,79
1,72
0,98
1,17
0,98
1,87
0,63
1,37
1,83
1,17
0,98
1,86
1,63
2,43
2,61
1,83
2,25
1,21
1,17
1,63
1,67
1,17
1,79
1,37
1,47
2,32

All Subjects
(40 subjects)
Aver.
8,35
8,05
8,13
7,88
7,65
7,55
7,48
7,40
7,58
7,38
7,38
7,38
7,23
7,18
7,33
7,23
6,88
6,93
7,13
7,08
6,87
6,73
6,69
6,38
6,63
6,60
6,50
6,64
6,58
6,60
6,48
6,53
6,44
6,40
6,33
6,33
6,36
6,30
6,35
6,15
6,23
6,28
6,28
6,05
6,08
6,03
6,13
6,15
5,90
6,03
5,90
5,85
5,88
5,85
5,85
5,75
5,65
5,60
5,67
5,60
5,58
5,70
5,55
5,33
5,38
5,40
5,13
4,98
5,08
5,00
5,15
5,18
4,85
5,08
4,70
4,78
4,75
5,00
4,70
4,85
4,48
4,43
4,23
3,95
3,85

Dev.
1,05
1,11
1,26
1,11
1,69
1,50
1,55
1,58
1,32
1,56
1,58
1,50
2,14
1,77
2,03
1,31
1,70
1,79
2,08
2,15
1,70
1,34
1,98
2,23
1,73
1,41
1,93
1,74
1,82
2,16
1,63
1,63
1,59
1,96
1,91
1,65
1,42
1,87
1,37
2,01
1,70
1,32
1,83
2,46
1,79
1,78
1,52
1,71
1,95
2,11
1,84
2,03
2,40
2,05
2,25
1,89
2,33
1,89
2,08
2,04
1,91
2,03
2,15
1,86
2,34
2,00
2,14
2,40
1,86
1,96
2,46
2,37
2,34
2,52
2,34
2,34
2,20
2,24
2,09
2,08
2,11
2,27
2,09
1,96
2,26


Posters
A PRIMITIVE EXECUTION MODEL
FOR HETEROGENEOUS MODELING

Frédéric Boulanger
Supélec – Département Informatique
3 rue Joliot-Curie, 91192 Gif-sur-Yvette cedex, France
Email: Frederic.Boulanger@supelec.fr

Guy Vidal-Naquet
Supélec and Université Paris-Sud
3 rue Joliot-Curie, 91192 Gif-sur-Yvette cedex, France
Email: Guy.Vidal-Naquet@supelec.fr

Keywords: Heterogeneous modeling, Models of computation, Execution models.

Abstract: Heterogeneous modeling is modeling using several modeling methods. Since many different modeling methods are used in different crafts, heterogeneous modeling is necessary to build a heterogeneous model of a system that takes the modeling habits of the designers into account.

A model of computation is a formal description of the behavioral aspect of a modeling method. It is the set of rules that allows to compute the behavior of a system by composing the behaviors of its components.

Heterogeneous modeling allows parts of the system to obey some rules while other parts obey other rules for the composition of their behaviors.

Computing the behavior of a system which is modeled using several models of computation can be difficult if the meaning of each model of computation, and what happens at their boundary, is not well defined. We propose an execution model that provides a framework of primitive operations that allow to express how a model of computation is interpreted in order to compute the behavior of a model of a system. When models of computation are “implemented” in this execution model, it becomes possible to specify exactly what is the meaning of the joint use of several models of computation in the model of a system.

1 CONTEXT

The design of most complex systems appeals to different crafts that are organized around sets of specific design methods, e.g. for industrial control or signal processing. These methods are adapted to specific aspects of a craft, and designers have a correct intuition of their semantics.

When integrating the different parts of a system, we generally translate the model of each part into a common low level formalism, or even into a common implementation language. By doing so, we lose all the information that tells how we went from the specification of the subsystem to its model. Therefore, when building the whole system, we cannot take advantage of the different choices of realization offered by the model, since they have been “frozen” in the low level implementation.

Another issue is that, when validating the behavior of the whole system, it will be difficult to find what should be changed in the model of a subsystem to insure a global property, since the low level implementation does not carry enough information about the design of the subsystem.

Heterogeneous modeling tries to overcome these issues by allowing to describe the whole system as a composition of subsystems that are designed according to different methods (Liu et al., 2003).

It does not provide a greater expressive power than other modeling techniques, but it allows the different teams that work on the design of a system to share a common model of the system, while using their own modeling techniques.

In the following, we present the actor paradigm for heterogeneous modeling and we propose a constructive method for computing the behavior of heterogeneous models. Projects like KerMeta (Fleurey et al., 2006) or Rosetta (Kong and Alexander, 2003) are related to our works, but focus on different objectives: KerMeta defines behaviors for the elements of the Meta-Object Facility (MOF) of the OMG, while Rosetta defines the combination of models of computation and uses a hierarchy of compatible models of computation. Our objective is to provide a framework in which any model of computation may be used to compute the interactions of components of a system. In this paper, we focus on the steps that are required to compute the behavior of a model of a system.
2 ACTORS, PORTS, RELATIONS

Our approach to modeling is based on the actor paradigm: a system is built from components named actors. Actors have properties and communicate through ports. Ports have properties and are linked by relations which also have properties. So, building a model of a system amounts to using some actors, to set their properties and the properties of their ports, and to build relations between the ports of the actors. The effective behavior of the model is obtained by interpreting the properties and the relations according to a model of computation.

Actors, ports, properties and relations are the elements of the abstract syntax of actor-oriented modeling. Models of computation are semantics for this abstract syntax: a model of computation is an interpretation of the relations between the ports of actors and the properties of these relations.

2.1 Roles of a Model of Computation

A model of computation allows to compute the observable behavior of a model of a system from the individual observable behaviors of its components. For instance, on figure 1, the MoC_{ext} model of computation computes the behavior of the top-level model that contains actors A and B. This model of computation is in charge of computing the status (availability of data and value) of B_{in} from the status of A_{out}. The status of A_{out} is determined from the status of A_{in} by the behavior of A. In the example, this behavior is described by a model that contains three actors and is governed by the MoC_{int} model of computation.

![Figure 1: Internal and external models of computation.](image)

MoC_{ext} is the “external model of computation” of actor A. It is the model of computation that combines its behavior with the behavior of other actors to compute the behavior of a model. MoC_{int} is the “internal model of computation” of A. It is the model that computes the behavior of A by combining the behaviors of the actors of the model of A.

Information is obtained by observing the output ports of actors, so the model of computation must tell how information goes from output to input ports, and when it is available on input ports. We can consider the first aspect (the propagation of data from output ports to input ports) as communication, and the second aspect (when data is available) as control or synchronization.

Communication consists generally in copying data from output ports to input ports, but synchronization can be more complex because it defines the type of causality used by the model of computation. In some models of computation, data produced on an output port is available immediately on all input ports that are in relation with that output port. In other models of computation, a notion of “tick” is used to relax causality: a data sample produced on an output port is available on all inputs ports that are in relation with it, but only after the next tick. In other models of computation, a notion of time is introduced to label data samples with a time-stamp which tells when they are available.

Communication and synchronization aspects must be described precisely to define a model of computation. There are actually very few tools or languages for describing these aspects. Most of the time, one has to implement a model of computation in a generic programming language.

3 EXECUTION MODEL

When the precise semantics of a model of computation is coded in a programming language, it is generally done in the context of a framework like Ptolemy II (Brooks et al., 2005) that provides support for the abstract syntax of the models. For such a framework to support an open set of models of computation, it must consider components as black boxes that compute the availability and values of data on input ports from the availability and values of data on output ports.

We define here a generic execution model that can be used with any model of computation. It relies on models of computation for determining in which order the actors of a model should be observed, and how the values on input ports are computed from the values observed on output ports. This execution model has matured from previous works (Boulanger et al., 2004; Feredj et al., 2004) based on the Ptolemy framework.

3.1 The Nature of an Execution

Before defining our generic execution model, we must define what is an execution of a model of a system. In order to keep our approach generic, we ignore the internal mechanisms of an actor, to focus on what and
when an actor produces data on its output ports, not how it produces it.

Therefore, in the following, we will never “trigger” the behavior of an actor, we will just observe its output ports. The behavior of an actor can occur any time – it can be a continuous process that runs during the whole execution of the model – but it must provide a coherent view each time its ports are observed. We use a stroboscopic effect to observe the actors of a model simultaneously in a series of snapshots. We consider that an execution of a model is a sequence of snapshots of the values available on the ports of the model. In a given snapshot, each port has a single defined value.

The nature of a snapshot depends on the model of computation. The execution model only tells the actors that a snapshot is going to be taken, and then asks them to approve the value of their ports as they appear on the snapshot. For an analogy, a photographer tells the actors “stay still”, and then asks them if they are pleased by the picture.

This definition of an execution implies that we are only interested in discrete behaviors. This is because our goal is not to describe behaviors, but to compute them. For instance, when we consider a model of a physical system, like a system of ordinary differential equations, we are not interested in finding properties of these ODEs but in computing the value of the outputs of the system at a discrete set of instants.

Since we are interested in observations only, each model of computation is insulated from the definition of the others, and there is no need to define the composition of any pair of models of computation.

For instance, if an actor is a sensor which acquires information from the external world, what is interesting is the result of the measure, not the mechanism for elaborating this measure.

### 3.2 Types of Actors

To define a generic execution model, we must consider the different ways with which an actor produces its outputs. **Strict actors** need to know the value of all their inputs to determine the value of all their outputs at once. With strict actors, a model cannot contain instantaneous causality loops because a strict actor cannot have an input that depends on the value of one of its outputs in the same snapshot. **Non-strict actors** can determine some of their outputs when they know the value of only some of their inputs. A delay is a non-strict actor: the value of its output depends only on its state, which in turn depends on the value that its input had in the previous snapshot. A logical OR gate is also a non-strict actor because its output is known as soon as one of its inputs is true.

When a model of computation supports non-strict actors, the values of the ports are determined iteratively. First, actors are provided with known inputs and they determine part of their outputs. These newly determined outputs allow to compute new values for inputs according to the model of computation. These newly determined inputs allow actors to determine more outputs, and so on until all the ports have a known value. With such models of computation, it is necessary to tell actors when new inputs become available.

Actors, independently of their strict or non-strict nature, may not agree with the value they have computed for their outputs for the current snapshot. For instance, consider a level-crossing detector. It produces an event when a signal crosses a threshold. If the signal is computed by numerical integration of differential equations, the integration step is adjusted so that the value of the signal is computed with a given precision. However, the measure may be refused if the integration step is too large for the temporal precision required on the event, and the snapshot will be recomputed again with a smaller integration step. A snapshot is considered valid when all the actors of the model agree with the value assigned to their ports.

### 3.3 Generic Execution Model

The taxonomy of actors presented above, and the fact that we are only interested in observations of the ports of actors, not in the activity of the actors, allows us to define a generic execution model that is capable of executing models that obey any model of computation. To attain such universality, we made as few assumptions about actors as possible, and we rely on an operational description of the model of computation to schedule observations and to compute the value and availability of data on input ports from the data available on output ports.

One can wonder at the previous sentence since we are used to outputs computed from inputs, not the reverse. The key is to consider that if actors produce their outputs from their inputs, the model of computation interprets the relations between ports to determine what is available on inputs ports from what is available on output ports.

We can now describe the steps of our execution model to compute a snapshot of the execution of a model of a system, and define the primitive operations that an actor must provide:

1. The **start_of_snapshot** operation is invoked on each actor of the model. In response to this invocation, an actor prepares for the snapshot. For instance, an actor that acquires information from the environment of the system (reading data from a file, sampling a sensor) should do it during this step.
2. the reset operation is invoked on each actor of the model. In response to this invocation, an actor resets its ports to the “unknown” state.

3. the update operation is invoked on the actors of the model returned by the schedule operation of the model of computation. In response to this invocation, an actor makes data available on its output ports. If the actor is strict, it makes data available on all its output ports. If it is non-strict, it makes data available only on the ports it can determine.

4. the operational description of the model of computation is used to compute the status (availability of data and value of the data) of the input ports from the status of the output ports.

5. if the model of computation determines that the snapshot is complete, go to step 6, else, go back to step 3. Steps 3 to 5 constitute an observation of the model.

6. the validate_snapshot operation is invoked on each actor of the model. In response to this operation, an actor considers the data available on its ports as definitive for this snapshot and tells whether it considers it as correct or not. If it does not validate the data, it should change some property of the model of computation (e.g. the integration step in our example with the level-crossing detector) so that a new computation of the snapshot will compute data that it may validate.

7. if all the actors of the model have validated the snapshot, go to step 8, else go back to step 2 to compute the snapshot again with the new parameters of the model of computation that have been set by the actors which have not validated the snapshot.

8. when all the actors of the model have validated the snapshot, the end_of_snapshot operation is invoked on each actor of the model. This operation tells actors that the snapshot is valid and that they can use the data available on their ports in their own activity or to update their internal state if any. Actors that provide data to the environment of the system (writing data to a file, driving an actuator) should do so during this step.

An actor should not update its internal state, change its activity or perform any operation that may have side effects on the environment between the start_of_snapshot and end_of_snapshot operations. The fact that a snapshot may be computed several times to converge toward a result that is accepted by all the actors must not be visible outside the model. For instance, if we consider our example of a level-crossing detector, it may be necessary to compute a snapshot several times before the integration step becomes small enough, but outside the model, only the last level-crossing event must be visible because it is the only one that has been considered as correct. For the same reason, new data should not be acquired during the computation of a valid snapshot.

3.4 Discussion on Steps

The overall structure of our execution model is shown on figure 2, with “actor operations” in rectangular boxes, “model of computation operations” in rounded corners boxes, and control choices in diamond shaped boxes.

The schedule operations of the model of computation determine which actors should be observed. These operations are executed each time new data becomes available: at the beginning of an observation (pre-schedule), in order to compute control from the inputs of the model and to determine which actors may produce observable outputs; after update (in-schedule) in order to handle data made available on the output ports of the actors; and at the end of the observation (post-schedule), in order to handle the outputs of the model.

Every model of computation must implement pre-schedule because this operation tells which actors will be observed in the current turn of the loop. The other two schedule operations may do nothing in models of computation where the control does not depend on data.

The first and last steps of the computation of a snapshot insulate the environment of the model from the internal changes that occur in the model during the computation of the observation of its ports. They also insulate the behavior of the actors of a model from the details of the computation of a snapshot of this model, since an actor is not allowed to update its internal state before the end_of_snapshot step. For instance, an actor should not count the number of times its reset, update or validate methods are invoked and make its future behavior depend on this count.
We can consider the `start_of_snapshot` step as “sample the external world”, and the `end_of_snapshot` as “update state, act on external world”. Between these two steps, actors must have a combinational behavior. This model is therefore very close to the synchronous sequential model where registers are loaded on the ticks of a clock with the results of combinational computations. In our execution model, the clock is the series of instants at which a snapshot exists. We do not need a more elaborate model of time at this level of the execution of a model of a system.

The `reset - validate` loop is crucial for heterogeneous modeling because it is the way by which the model of computation that is used for the internal model of an actor can influence the model of computation in which the actor is used. In a model of a system, actors are just observed, and the observations are combined by a model of computation to build an observation of the model. However, the behavior of each actor can also be described by a model of the actor (the actor is considered as a subsystem), and the model of computation used to model the behavior of the actor can be different from the first model of computation. We call “external model of computation” the model of computation that is used to combine the behavior of an actor with the behavior of other actors, and “internal model of computation” the model of computation that is used to describe the internal behavior of an actor.

In order to avoid to compute all the possible combinations of models of computation, we hide the internal model of computation to the eyes of the external one. Since the external model of computation “decides” when the ports of an actor are observed, the internal model of computation would have no control on the computation of the snapshot if it could not refuse a computation by making the actor return `false` to the `validate` request.

The “observe” loop implements a well-known technique to compute the behavior of a model as a fixed-point. It is implemented in Ptolemy II by the `prefire`, `fire` and `postfire` methods. `prefire` is the equivalent of `start_of_snapshot` in our execution model, `fire` is equivalent to `update` and `postfire` to `end_of_snapshot`. However, we chose different names since there is no `reset - validate` loop in the general execution model of Ptolemy II (even if such a validation steps exists in the “Continuous Time” model of computation), and the names of these methods denote the activation of a behavior. Our execution model deals only with observations and we do not limit the behavior of actors to the body of a `fire` method.

3.5 Allowing Heterogeneity

The execution model we have just presented here uses only one model of computation, so one may wonder how heterogeneous models are handled. Our approach of heterogeneity is the same as the hierarchical approach used in Ptolemy (Eker et al., 2002), and our execution model does not depend on the models of computation used to compute the behavior of the actors of a model. It is therefore possible to define the behavior of actors using internal models of computation that differ from their external model of computation.

An issue still subsists: how data produced according to the internal model of computation of an actor will be interpreted in the context of its external model of computation? The behavior of an actor may be expressed using properties that have no meaning in the external model of computation. For instance, an actor may produce time-stamped data samples because its behavior is defined using a timed model of computation, and these samples may be read in a model of computation that has no notion of time. In this case, the series of timed-stamped data samples can be viewed as a sequence of data samples just by discarding the time-stamps, but in the reverse case, when data with no time-stamp is produced in a timed model of computation, a time-stamp must be created for each data sample, and this requires additional information.

Our position is that there is no automatic way to convert data (or control) from a model of computation to another. Often, there are standard ways of adapting the semantics of two models of computation (for instance, periodic sampling can be used to go from continuous time to synchronous data-flow), but such transformations should not be “hard-coded” in the modeling framework nor applied implicitly to an heterogeneous model. The reasons for this are:

- implicit transformations are framework-dependent. This means that the same model could adopt different behaviors when executed in different modeling frameworks;
- several transformations between two models of computation may exist (for instance, when going from discrete to continuous time, it is possible to hold the last value, or to use linear or more complex interpolation). The choice of a transformation is part of the design of the system, and it should therefore appear explicitly in the model;
- even when there is only one possible transformation between two models of computation, using this transformation and setting its parameters is a design choice, and it should appear in the model of the system, with the same importance as the models of computations.
The main problem with such transformations is that if they are implemented as actors, these actors appear either in the internal or in the external model of computation. Both ways are wrong since they break modularity: if the internal model of an actor contains actors to adapt data to its external model of computation, this internal model depends on the external model of computation. This means that the design of an actor depends on the context in which it will be used. The same problem occurs when the adapting actors are placed in the external model of computation. In (Feredj et al., 2004), we presented a model for domain-polymorph components that allows the adaptation between two models of computation to be done at the interface between the models. This approach turns the adaptation between the semantics of the internal and external models of computation into a property of the edge of the actor.

A last issue is that it is sometimes necessary to define actors which obey several models of computation. For instance, a sampler has a continuous input, a discrete event input (the sampling clock) and a data-flow output (the sequence of samples). A level-crossing detector has a continuous or sampled input and a discrete event output. Such actors cannot be handled directly in our execution model because only one model of computation is allowed in the model of a system. However, we have shown in (Boulanger et al., 2004) that a flat heterogeneous model, i.e., a model that uses several models of computation at the same level of its hierarchy, can be rewritten automatically into a hierarchical model by projecting heterogeneous actors on the models of computation they use. One may also consider that such behaviors should not be modeled as actors but as transformations between models of computation, and considered as properties of the edge of models, as evoked earlier.

4 CONCLUSION

We have presented the roles of a model of computation and the different kinds of actors it should be able to manage, and then an execution model which, by making as few assumptions as possible about actors, is able to execute models that obey any model of computation. Our works on the integration of the reactive synchronous approach into object-oriented programming and on the adaptation between models of computation in the Ptolemy framework make us quite confident in the universality of this model. By considering only observations on the ports of actors, and not the activity of actors, this execution model can safely ignore what happens at lower levels of the hierarchy of a model. This allows the use of different models of computation at different level of the hierarchy of a model of a system. Moreover, by allowing an actor to veto the result of the computation of a snapshot of the model, this execution model allows inner models of computation to interact with the outer models, in addition to the usual control that the outer model has on the inner models of computation.

This execution model requires that a model of computation is able to provide a schedule of the actors of a model, and to propagate the data observed on the output ports toward the input ports. These two operations can be complex, and are, for the moment, implemented using generic programming languages like Java or C++. Our goal is to describe them formally using either an extended version of the Object Constraint Language (OCL) or the Action Language of UML 2, with a mathematical foundation for the in order, particularly, to define transformations from a model of computation to another and to handle heterogeneity in a more generic way.

REFERENCES


INTRODUCTION TO CHARACTERIZATION OF MONITORS FOR TESTING SAFETY-CRITICAL SOFTWARE

Christian Di Biagio, Guido Pennella
MBDA-Italy SpA, Via Tiburtina, Roma, Italy
<christian.di-biagio, guido.pennella>@mbda.it

Anna Lomartire
Centro di Calcolo e Documentazione, Università degli Studi di Roma “Tor Vergata”, Via O. Raimondo, Roma, Italy
annal@ccd.uniroma2.it

Giovanni Cantone
Dip. di Informatica, Sistemi e Produzione, Università degli Studi di Roma “Tor Vergata”, Via O. Raimondo, Roma, Italy
cantone@uniroma2.it

Keywords: Software engineering, Distributed and parallel systems, Hard real-time Systems, Performance-measurement tools.

Abstract: The goal of this paper is to characterize software technologies to test hard real-time software by focusing on measurement of CPU and memory loads, performance monitoring of processes and their threads, intrusiveness, and some other key features and capabilities, in the context of the Italian branch of a multinational organization, which works in the domain of safety-critical systems, from the points of view of the project managers of such an organization, on one side, and the applied researcher, on the other side. The paper first sketches on the state of the art in the field of testing technologies for safety-critical systems, then presents a characterization model, which is based on goals of the reference company, and then applies that model to major testing tools available.

1 INTRODUCTION

The development of safety critical software in industrial settings is usually influenced by user non-functional requirements that concern the load (e.g., the usage of the CPU and Memory in a period), which is specified not exceed a fixed level, of any computing node in a certain scenario.

Before designing safety-critical or mission-critical real-time systems, a specification of the required behaviour of the system should be produced and reviewed by domain experts. As the implementation advances, eventually it completes, the system is thoroughly tested to be confident that it behaves correctly. In fact, the concept of software verification and validation was eventually extended up to include quality assurance for new digitalized safety-critical systems (EPRI, 1994).

The test of the system’s temporal behaviours seems best done when using a monitor, i.e. a system able to observe and analyze behaviours shown by another remote system (a.k.a.: the “target”). Several authors (e.g. (Tsai, 1995) also suggested that it is useful and practical using monitors to analyze the behaviour of a real-time system. Such a monitor could be used either as an “oracle” (Weyuker, 1982), which reports true values during system testing, or, for a limited class of systems, as a “supervisor” (Simser, 1996), which detect and report system failures during operation.

In safety-critical applications, the system should be monitored by an independent safety system to ensure continued correct behaviour. To achieve these goals, there must be a means for quickly determining if the observed behaviour is acceptable or not; this can be quite difficult for complex real-time systems. In other words, because software practitioners cannot diagnose, troubleshoot, and resolve every component affecting a critical software performance by using just manual methods, the consequent question is: To what extend the testing technology that the market provides is able to give practitioners help in verifying the temporal
behaviour of their safety-critical software, seeing a problem in real time, drilling down and resolving it fast?

The reference company for this paper – the Italian branch of a multinational organization in the field of safety & mission critical software – asked us that question when the need emerged from her production lines for a test-suite to validate internal software products. In fact, her project managers were unsatisfied with their testing technologies and approaches, and were addressing their processes and products for quality improvement. Of course they were not looking just for one more test tool, but for a technology able to meet their improvement goals.

This paper is concerned with answering that aforementioned question. In GQM terms (Basili, 1994): The goals are to characterize testing technologies by focusing on measurement of CPU and memory loads, performance monitoring of distributed heterogeneous processes and their threads, intrusiveness, and other key features and capabilities, in the context of a multinational organization for the domain of individual/social life-critical systems, from the points of view of the project manager and the applied researcher, respectively.

In the remaining of the present paper, Section 2 surveys on, and analyzes features provided (or non-provided!) by the major tools for monitoring the testing of hard real-time software. Section 3 collects the results from analysis above, and Section 4 evaluates those results. Section 5 presents some conclusions and points to future research.

2 MAJOR TECHNOLOGY AVAILABLE

In the present section we focus our attention on the most known system-load monitoring tools. These, to the best of our knowledge, are:

- Quest SpotLight™ (Quest SpotLight, 2006)
- TOP (William Lefebre’s Top, 2006)
- Solaris Performance Meter™ (Solaris Performance Meter, 2006).

Let us note additionally that, again to the best of our knowledge, Quest SpotLight™ and (William Lefebre’s) Top™ are the most used tools for Unix standard OS.

2.1 Quest SpotLight™

Based on official documentation (Quest SpotLight, 2006), this tool graphically displays the real-time flow of data within MS Windows OS, so enabling the user to watch and respond to problems before they become a major concern. Key Features are: (i) Graphical, actionable diagnostic console, which combines data from multiple sources.

![Figure 1: A Quest SpotLight™ output.](image)

(ii) Automatic calibration: the tool offers a calibration process that automatically sets a baseline of normal activity and thresholds for each system. (iii) Detailed process tracking capabilities: the tool displays up to 24 hours of historical information about specific processes including CPU usage, number of threads, handles, and page faults. (iv) Event Log tracking: the tool alerts the user whenever specific or general event log entries have been generated on the servers being viewed.

Figure 1 shows an output from Quest SpotLight™.

2.2 TOP

Based on official documentation William Lefebre’s Top, 2006), the system utility Top provides a continuous, real-time look at the system's consumption of memory and CPU resources. It lists the most consumptive process first, so finding that process that is gobbling machine resources is relatively easy. Top also displays: the total operation time for the system since the last reboot; load averages; process counts for various states; the percentage of CPU time broken down between user, system, nice, and idle; memory and swap space usage; as well as the list of the processes using the largest amount of the machine resources. Figure 2 shows a sample output from Top.

2.3 Solaris Performance Meter™

This tool is frequently used to monitor activity and performance on a workstation. Several performance parameters such as CPU utilization, disk activity,
network packets, and the like, can be displayed graphically in a customizable window.

Last pid: 22336; load averages: 0.12, 0.11, 0.09 11:39:58
80 processes: 73 sleeping, 6 zombie, 1 on cpu

Memory: 256M real, 90M free, 34M swap in use, 351M swap free

PID USERNAME THR PRI NICE SIZE   RES STATE TIME    CPU COMMAND
21440 root       1  35   -3   12M   11M sleep   0:20  1.74% ncftpd
22336 mortimer   1  -7    0 1368K 1264K cpu/0   0:00  0.63% top
21075 root       1  34   -3 1832K 1456K sleep   0:16  0.33% ncftpd
127 msql       1 -25    0 1640K  936K sleep 254:03  0.18% msql2d
22305 www        1  33    0 2728K  2112K sleep 22404  0:00  0.04% httpd
22308 www        1  33    0 2728K  2112K sleep 22304  0:00  0.04% httpd
22296 www        1  33    0 2728K  2112K sleep 22308  0:00  0.02% httpd
22292 www        1  33    0 2728K  2112K sleep 22306  0:00  0.02% httpd
22302 www        1  33    0 2656K

Figure 2: A sample output from Top.

Solaris Performance Meter™ users can monitor performance of local or remote hosts, set up colour-coded activity thresholds to raise warns in case of exceptional performance, and log the samples to a file.

Figure 3 shows a typical output from Solaris Performance Meter™ (Solaris Performance Meter, 2006).

3 RESULTS

Let T1, T2, and T3 denote, in any order, the three tools sketched by Section 2.3 above (it is not our role to advertise or counter-advertise tools; so we do not map comments and tools).

Table 1 synthesizes on the characteristics of T1, T2, and T3, in the perspective of a model of ideal technology that we constructed on needs placed by testing professionals at our reference company. This model was based on cost and 17 features, which are synthetically presented in Table 1, Column 1. These features relate to tools capabilities, including: to cope with heterogeneous targets, CPU and memory load for system, processes and threads, data persistency, tailorability, non-intrusiveness, ability to cope with distributed systems and multi-platforms (Di Biagio, 2006).

Because many of the values in Table 1 are null, we renounced to assign weights to features and compute an indicator for each of the shown tools.

Concerning T1, it outputs data on, and continually refresh, a shell. While T1 is sufficiently non-intrusive, it resides on the target system, where repositories and graphic and statistical analysis packages are usually not allowed. This means that there is no support for: (i) Monitoring different targets at the same time, in order to compare them in real time. (ii) Reviewing tests and DB repository; (iii) Tailoring to minimize intrusiveness. (iv) Thread-monitoring to observe the behaviour of developed products. In our view, the main lack of T1 concerns its architecture, which is not suitable (Simser, 1996).

Concerning T2, main lacks regard again its architecture, in our view. In fact, T2 accesses the target system through TCP/IP over Ethernet, where no sensor is installed. This means that data acquisition is system-call enacted (i.e. the OS
command “ps”). As a consequence, measurements are strongly intrusive (up to 60% of CPU during acquisition, in our experience). Moreover, there is no support for: (i) Monitoring different targets at the same time, in order to compare them in real time. (ii) Reviewing tests for future reuse, and DB repository. (iii) Tailoring to minimize intrusiveness. (iv) Process and thread monitoring.

Concerning T3, in our view, its major limit is the absence of supports for: (i) Monitoring different targets at the same time, in order to compare them in real time. (ii) Reviewing tests for future reuse, and DB repository. (iii) Tailoring to minimize intrusiveness. (iv) Process and thread monitoring. (v) Solaris is the only OS that T3 supports.

4 DISCUSSION

All the major tools for monitoring hard real-time software seems to present substantial limits with respect to the ideal technology of our reference company (see Table 1).

T3 seems too far from that ideal: in fact, multiple monitoring (F1), data storage (F4), tailoring (F5), and process monitoring (F11 .. F14) are not supported at all. Concerning T1 and T2, while at a first view they seem to match many of the features and capabilities that our ideal model requires, they lost such a primacy when we look deeper for their intrusiveness (F6): in fact, this is one of the most important aspect in safety critical software. T1 seems to best fit many other required features and capabilities, Anyway, it does not support tailoring (F5), data storage (F4), distributed architecture (F7), threads monitoring (F13, F14).

Overall, all those tools show a main limit: none of them provides what we called with Sensor (F17), i.e. a module built right for acquiring and sending-out data by using negligible resources and time. Of course, they carry out those activities, but in different, often broad, ways. In particular: (i) T1 is not so much intrusive, and sensitive data are continually refreshed. However, it resides on the target, which is expected to be not in charge of providing utility functions. (ii) T2 accesses the target system through TCP/IP, where no sensor is installed: because of the consequent usage of system calls, the tool is strongly intrusive. (iii) T3 is non-intrusive, but the set of data it is able to acquire is very limited.

As a conclusive remark, the real trouble with traded tools seems to be that they assume the point of view of the “System Administrator”, so answering questions like: “What is the problem”, “Where is the problem”, “Who generated the problem”.

5 CONCLUSION AND FUTURE WORK

We have presented a model, which is based on the quality improvement goals of the reference organization for this paper, and aimed to characterize technologies for testing time-properties of safety-critical software. We have also presented results from the application of that model to three major tools for monitoring hard real-time software during test sessions. Based on those results, it seems that the technology provided by the market does not meet sufficiently the needs of our reference company. Management of that company is hence invited to evaluate the chances they have to develop in house their ideal technology for something like this.

REFERENCES

SOLARIS PERFORMANCE METER™ 2.0.0 http://docsun.cites.uiuc.edu/sun_docs/C/solaris_9/SUNWber/CDEUG/p125.html
TOP- William LeFebvre’s http://www.uwsg.iu.edu/UAU/system/top.html
MODELLING THE UNEXPECTED BEHAVIOURS OF EMBEDDED SOFTWARE USING UML SEQUENCE DIAGRAMS

Hee-jin Lee, In-Gwon Song, Sang-Uk Jeon, Doo-Hwan Bae
Department of EE and CS, KAIST, Daejeon, Republic of Korea
leehj, igsong, sujeon, bae@se.kaist.ac.kr

Jang-Eui Hong
School of Electrical & Computer Engineering, Chungbuk National University, Cheongju, Korea
jehong@chungbuk.ac.kr

Keywords: Embedded software, Exceptional behaviour modelling, UML Sequence diagram.

Abstract: Real-time and embedded systems may be left on unexpected states because system’s user can generate some incident events in various conditions. Although the UML 2.0 sequence diagrams recently incorporate several modelling features for embedded software, they have some difficulties to depict unexpected behaviours of embedded software conveniently. In this paper, we propose some extensions to UML 2.0 sequence diagrams to model unexpected behaviours of embedded software. We newly introduce notations to describe exceptions and interrupts. Our new extensions make the sequence diagrams simple and easy to read in describing such unexpected behaviours. These features are explained and proved with an example of call-setup procedure of CDMA mobile phone.

1 INTRODUCTION

The development of embedded software is getting more attention by researchers and developers as the size and complexity of embedded software increase. Embedded software has special requirements on timing, performance, and device interface. Moreover, there are some considerations in embedded software modelling as follows:

- Embedded software has timing constraints in the aspects of soft real-time or hard real-time.
- Events from input and to output are limited to specific resources.
- It is impossible to forecast when the input events from external users occur.

Embedded software is a reactive system. Depending on the input events, adequate behaviour should be performed. There are mainly two types of embedded software behaviours. First, predefined behaviour is executed by expected inputs. Second, unexpected or abnormal behaviour occurs by undefined inputs which are from users or environments unexpectedly. Not to mention the importance of the first case, the second case is also important in embedded system, because unexpected input may cause the system halt or do harm. Therefore, the reactions for unexpected inputs as well as normal or defined inputs should be considered in the modelling of embedded software.

It is known that sequence diagrams in UML are adequate to model the dynamic system behaviours. The latest release of it, version 2.0, incorporates several notations for the modelling of embedded software. Although the representation of unexpected behaviours such as interrupts or exceptions in standard sequence diagrams is possible, the sequence diagrams describing those behaviours become complicated and intricate. Thus, we propose extended notations with the definition of their syntaxes and semantics to avoid unreadable sequence diagrams in describing unexpected behaviours. We also explain and show the effectiveness of the unexpected behaviours modelling in the aspects of readability, abstraction, and simplicity.

The rest of this paper is organized as follows.: Section 2 explains the characteristics and the usefulness of sequence diagrams and Section 3 describes our extensions of sequence diagrams for embedded software. Section 4 compares our
extended sequence diagrams and MSCs with example scenarios. Section 5 addresses related works. Finally, Section 6 concludes the paper and discusses about future work.

2 BACKGROUND

When describing the dynamic behaviours of a system with UML, we use sequence diagrams, state machine diagrams, and activity diagrams (Douglass 2004). The activity diagram is a model to describe a business process or a method of a class. The statemachine diagram describes the states and the actions of each object in its lifetime. Although the activity and statemachine diagrams are capable of modelling the dynamic behaviours of the system, the sequence diagrams seem to be more practical for software engineers in industry to describe the behaviours of embedded systems. It is because sequence diagrams are suitable to draw models from requirements straightforwardly and easy to understand for developers. Also, they describe the global interactions as well as the partial behaviours between objects. Due to the intuitiveness, sequence diagrams are generally preferred to the statemachine diagrams for describing software behaviours.

In addition to the usefulness of sequence diagrams as described above, UML 2.0 sequence diagrams become more expressive in system behaviour modelling by consolidating the inline expressions and the time concepts of MSCs (ITU 1999, Mauw 2000, Damm 2001, Haugen 2004, and Haugen 2001).

Even though the expressive power of sequence diagrams is enhanced, the modelling of unexpected behaviours often causes redundancies of other behaviours and makes sequence diagrams unreadable. Unexpected behaviours such as interrupts and exceptions are generally controlled by system calls of the operating system. However, we focus on special situations that those unexpected behaviours should be handled in application level or in bare machine which has no operating system.

From these motivations, we realize that the UML 2.0 sequence diagrams should be extended to describe unexpected behaviours of embedded software.

3 MORE FEATURES IN SEQUENCE DIAGRAMS

Exceptions and interrupts occur frequently in the operations of embedded software. Therefore, they should be represented in sequence diagrams to depict unexpected behaviours in a view of user-defined event modelling.

Figure 1: UML profile for extended sequence diagrams.

We extend the combined fragments of sequence diagrams to describe the handling of exceptions and interrupts. Extended interaction operators are ‘try’ for an exception handling and ‘interrupt’ for an interrupt handling. An exception scenario is recognized as an unsuccessful scenario. It occurs when certain constraints are not satisfied. Generally, an interrupt is controlled by system calls of the operating system. However, we define an interrupt as one of the events that occurs in the scenarios of application level. When an exception or an interrupt occurs, the execution of the current scenario is stopped and a handling scenario is executed. However, there are differences between the handlings of two unexpected scenarios. The occurrence of an exception is dependent on current executing action. However, an interrupt occurs regardless of the current action.

Figure 1 shows an UML profile (Eriksson 2003) for our extension of exceptions and interrupts in embedded software. A stereotype ‘Catch’ and a class ‘Exception’ are added for ‘try’ interaction operator. The stereotype ‘Catch’ is a kind of the stereotype ‘Message’. The inherited classes from the class ‘Exception’ are selectively used in sequence diagrams according to their properties.
3.1 Exception Handling Fragment

UML 2.0 sequence diagrams do not provide notations for specifying or handling exceptions. Therefore, we introduce a fragment ‘try’ which handles exceptional behaviour. The processing of an exception is considered in two aspects: a raising and a handling (Storrle 2004).

The exception raising is described with three parts: the trigger, the scope of readiness, and the scope of preemption (Storrle 2004). Under our notation, the trigger is one of ‘DurationConstraint’, ‘TimeConstraint’ and ‘StateInvariant’. The scope of readiness is a place or a point that the exception can arise, and the scope of preemption is the first operand of ‘try’ fragment.

Table 1: Symbols used in interaction operator ‘try’.

<table>
<thead>
<tr>
<th>FEATURE</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragment ‘try’</td>
<td>&lt;&lt;ExtendedOp&gt;&gt; try</td>
</tr>
<tr>
<td></td>
<td>[DurationConstraint, TimeConstraint, StateInvariant]</td>
</tr>
<tr>
<td>Catch message</td>
<td>&lt;&lt;Catch&gt;&gt; Catch(e1)</td>
</tr>
</tbody>
</table>

An exception can occur during the execution of normal scenarios within the ‘try’ fragment. When the exception occurs, an appropriate handling scenario will be performed. Symbols used for an exception handling ‘try’ are shown in Table 1.

- Interaction Operator ‘try’: The combined fragment ‘try’ consists of two or more fragments. The first fragment describes a scenario in which exceptions may occur. Each fragment of the rest describes the handling scenario of each of those exceptions.
- Catch message: Catch message with stereotype ‘Catch’ recognizes Exception ‘e1’ occurs in the first fragment.

There are three kinds of exception types; DurationConstraintException (DCE), TimeConstraintException (TCE) and StateInvariantException (SIE) (Goodenough 1975, Strohmeier 2001).

- DCE is on the handling of duration exception. If an event is not progressed within a predefined duration, DCE will occur.
- When an event does not happen at a particular time, TCE occurs.
- SIE occurs when an invariant constraint is not satisfied.

Figure 2 shows an example scenario of playing movie files. Object ‘FrameDecoder’ decodes movie files and sends the decoded data to ‘DisplayDevice’ object. If the decoding is not completed within a certain duration, a DCE exception will occur. The bottom fragment in Figure 2 shows the handling of such exception.

Figure 2: An example scenario of handling an exception.

3.2 Interrupt Handling Fragment

Although UML 2.0 sequence diagrams support the representation of the interruptible behaviour, we propose new notations to reduce the complexity of models that handle interrupts. When modelling the unexpected behaviour – i.e., an interrupt – using the existing UML sequence diagrams, many diagrams should be drawn. Thus we introduce an operator ‘interrupt’ which describes interruptible behaviours. Symbols used for interrupt handling are described in Table 2.

Table 2: Symbols used in interaction operator ‘interrupt’.

<table>
<thead>
<tr>
<th>FEATURE</th>
<th>SYMBOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragment ‘interrupt’</td>
<td>&lt;&lt;ExtendedOp&gt;&gt; interrupt</td>
</tr>
<tr>
<td>Interrupt signal</td>
<td>return</td>
</tr>
</tbody>
</table>

- InteractionOperator ‘interrupt’: The combined fragment ‘interrupt’ consists of two or more fragments. The first one describes a scenario that is interruptible by some interrupt messages. The others describe the handling scenarios for those interrupt messages.
- Interrupt signal: The message which is placed in a dotted long hexagon represents an interrupt message.
Return message: After receiving an interrupt signal, the original scenario is paused. The return message makes the paused operations resumed. If there is no return message, the original scenario is not resumed.

If an interrupt message arrives, the execution of a normal scenario stops and the execution control flow moves to an interrupt handling region to process the interrupt signal. Figure 3 shows an example scenario of playing movie files with an interrupt. It describes a scenario that ‘fast forward’ or ‘rewind’ button is pressed unexpectedly while the movie is playing. If the ‘rewind’ button is pressed, the execution of “Playing movie” interaction stops and the bottom fragment is executed.

Figure 3: An example scenario with interrupt handling.

In UML 2.0 sequence diagrams, interrupts could be described using fragment ‘alt’ (OMG 2004). Since the modeler does not know exactly when an interrupt would occur, he/she should put the ‘alt’ fragment into every single message. If there are more than one interrupt, the number of the ‘alt’ fragments in the sequence diagrams is increased as multiplied by the number of interrupts. For example, if there is a scenario that contains 20 messages and 5 interrupts, then 100 ‘alt’ fragments would be shown in the sequence diagrams.

4 COMPARISON OF MSC AND SEQUENCE DIAGRAMS

In this section, we compare our extended sequence diagrams with MSCs and UML 2.0 sequence diagrams through an example scenario of a mobile phone.

A Scenario of Mobile Phone
1. When there is a phone call, the caller’s information is shown and the bell is ringing.
2. When the bell is ringing, the user can answer the phone by pressing the call button.
3. The user can communicate with a peer through a speaker and microphone.
4. If the user does not answer the phone after 15 seconds ringing, it will stop transmission.
5. If the user or peer presses a stop button, the phone call is stopped.

Figure 4 shows the MSCs for a part of the scenario. Expression ‘exc’ is used to describe which exception occurs and how the exception is handled.

In this case, the exception is that the user does not press a button within 15 seconds after setting the ‘BellTimer’. If ‘BellTimer’ is timed out, ‘Stopping-Transmission’ scenario is performed as an exception handling.

Figure 5 shows the MSCs that describes the whole steps of the scenario. An external event, hanging the phone, is regarded as an interrupt signal. MSCs do not have any notation for interrupt handling. We use ‘exc’ expression to describe the interrupt. If the user or peer presses a stop button then the phone call is stopped. After pressing a stop button, as an interrupt, designated handling scenario is executed. Since it is not possible to know when the user hangs the phone, the ‘exc’ expression should be located after every message. It makes the model difficult to read and hard to understand.

With our extended notations, the handling of interrupts can be described in one sequence diagram as shown in Figure 6, which describes the above scenario. The ‘try’ fragment in the figure represents the exception handling scenario that should be executed when duration-constraint is violated. In addition, the interrupt scenario that can be occurred by user is described by ‘interrupt’ fragment.
surrounding the whole behaviours. In the extended sequence diagrams, the interrupt handling fragments do not need to be located on every pair of messages like Figure 5. The two extended notations can make the sequence diagrams simple and help understand the behaviours of the model easily.

With our extensions of sequence diagrams, we model the following four example scenarios:

1. ATM(Automated Teller Machine) scenario
2. Call signalling scenario with one interrupt in mobile phone
3. Call signalling scenario with two interrupt in mobile phone
4. Simple message editing scenario with ‘loop’ fragment in mobile phone.

In the first scenario, an interrupt occurs by the customer pressing a cancel button under normal operation. The second scenario is in case of the occurrence of an interrupt by hanging up the phone call by receiver. The third scenario is that the phone call is hung up by receiver of caller. The last scenario is in case of pressing OK button as an interrupt while a simple message is editing within 50 characters.

For the above four scenarios, we summarize the modelling results as shown in Table 3.

**Table 3: Example scenario modelling results.**

<table>
<thead>
<tr>
<th>no.</th>
<th>Number of messages (generated by user)</th>
<th>Number of UML sequence diagrams</th>
<th>Number of extended sequence diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>49</td>
<td>1</td>
</tr>
</tbody>
</table>

From the Table 3, we observed that our extended sequence diagrams provide some benefits in aspects of simplicity, understandability, and intuitiveness when describing unexpected behaviours. Also it can reduce the effort of the modelling dynamic behaviours in embedded software (Lee 2006).

### 5 RELATED WORK

Huget (Huget 2003) had introduced several extensions to the sequence diagrams of Agent UML, which is an UML extension for the interaction protocol domains. He had presented a notation for handling exceptions, a fragment named ‘exception’. However, the way of handling the exceptions was not mentioned. In our approach, we can describe the...
handling of exceptions as well as when they occur.

In UML, a ‘Signal’ is a metaclass defined as a specification of an asynchronous stimulus communicated between instances. An ‘exception’ is a special ‘Signal’ occurring with fault stimulus such as the violation of a preconditional or range invariant (OMG 1998). Douglass (Douglass 1999) had suggested the extended sequence diagrams that represent an exception handling. From his suggestion, a message stereotyped with ‘exception’ represents exceptional behaviours in embedded software. The exception message is limited to express negative scenario exception only.

6 CONCLUSION

In this paper, we presented an approach to extending UML 2.0 sequence diagrams to model unexpected behaviours of embedded software. Based on the profile, we added modelling notations into UML 2.0 sequence diagrams in order to describe unexpected behaviours in embedded software. Interrupts and exceptions frequently occur under the operation of embedded software. To model such unexpected behaviours, we used new interaction operators ‘try’ and ‘interrupt’ for handling exceptions and interrupts. The extensions in this paper help modelers design embedded software clearly, intuitively, and correctly.

There are some features to be considered. Interrupts and exceptions could be lost during the occurrences of other interrupts and exceptions. They should be handled during other events. However, our extensions could not cover those. It should be controlled or handled by operating the system level.

Our final goal is the application of our extensions to embedded software modelling for multi-processor SoC platform. Sequence diagrams for a multiprocessor system are more complex than those of a single processor system. We are under research about the modelling of unexpected behaviours of embedded software that are executed on multi-processor system.

ACKNOWLEDGEMENTS

This work was supported in part by IT Leading R&D Support Project funded by Ministry of Information and Communication, Republic of Korea and support program supervised by the IITA(Institute of Information Technology Assessment).

REFERENCES


ICSOFT 2006 - INTERNATIONAL CONFERENCE ON SOFTWARE AND DATA TECHNOLOGIES

262
VIEWPOINT FOR MAINTAINING UML MODELS AGAINST APPLICATION CHANGES

Walter Cazzola  
Department of Informatics and Communication  
Università degli Studi di Milano  
cazzola@dico.unimi.it

Ahmed Ghoneim and Gunter Saake  
Institute für Technische und Betriebliche Informationssysteme  
Otto-von-Guericke-Universität Magdeburg  
{ghoneim|saake}@iti.cs.uni-magdeburg.de

Keywords: Software evolution, automatic evolution and documentation, design and code coherence, UML.

Abstract: The urgency that characterizes many requests for evolution forces the system administrators/developers of directly adapting the system without passing through the adaptation of its design. This creates a gap between the design information and the system it describes. The existing design models provide a static and often outdated snapshot of the system unrespectful of the system changes. Software developers spend a lot of time evolving the system and then on updating the design information according to the evolution of the system. To this respect, we present an approach to automatically keep the design information (UML diagrams in our case) updated when the system evolves. The UML diagrams are bound to the application and all the changes to it are reflected to the diagrams as well.

1 INTRODUCTION

Software systems are expecting for a mechanisms to face changes in their environment and be able to self-adapt their code and design models when unanticipated events occur. The UML is de facto the standard (graphical) language used during the design process, therefore we consider its diagrams as a good representation for the system design (Booch et al., 1999). Dynamic events are hard to be captured at design-time whereas their occurrence surely affects also the design information. This problem forces a redesigning of the software systems when changes occur.

The urgency that characterizes many requests for evolution forces the system administrators/developers of directly adapting the system without passing through the adaptation of its design. This creates a gap between the design information and the system it describes (Cazzola et al., 2005). The existing design models provide a static and often outdated snapshot of the system unrespectful of the system changes. Software developers spend a lot of time evolving the system and then on updating the design information according to the system evolution.

Usually, software systems are described and documented by a set of design models. Evolving or redesigning these models to match the changes to the requirements and then updated the code requires a lot of time and efforts clashing with the urgency constraint. Instead, due to the pressing urgency, the developer has to directly adapt the system code and only successively the designer modifies the original design information according to the changes done by the developer. A post evolution updating of the design information from the evolved code is difficult, prone to erroneous interpretations and often comes too late to be adequate.

The challenges is to produce a framework that is able to adapt itself and keep updated its design information. We propose an approach that permits of adapting the design information of the evolving software system without requiring the work of the designer directly after the code adaptation. What we introduce is another point of view to maintain design information of the software system: the design and the code must evolve together. The evolution of the system is carried out by scripts (evolutionary rules) that evolve both the code and design information filling the gap between the two representations. The developer steers the design and code evolution through the implementation of the maintaining rules that describe how the changes to the requirements affect the system. Of course to support this approach is necessary an underlying middleware (the design information maintainer) that allows to interact with the design information as well as with the code. This paper focus on this aspect.
The rest of the paper is structured as follows: Section 2, describes the design information maintainer. Section 3, describes in more details the evolution of the design information through an example. Finally, in the Sections 4 and 5 we survey some related work, draw our conclusions and present some future work.

2 DESIGN INFORMATION MAINTAINER

The design information maintainer is logically divided in three layers: the design information layer, the intermediate-centric layer and the developer-centric layer.

The design information layer consists of the design models of the software systems in form of UML diagrams (and their internal representation XMI schemas). The intermediate-centric layer is responsible of observing and manipulating the XML of the design models after the directives of the developer-centric layer. This layer is responsible for implementing the new requirements in a set of maintenance rules that will adapt code and indirectly also the design information.

2.1 Design Information Layer

The design information is the central concept for documenting a software system and it plays also a relevant role in the system maintenance. The UML is the considered formalism for representing the design information. The framework will have a dual approach to the design manipulation: i) the maintenance rules work on the diagrams but ii) the manipulation will take place on the XMI schemas.

Most of the available UML tools provide the ability of describing the system by drawing UML diagrams and storing them as XMI schemas. In general, the designer will directly use these tools to manipulate the UML diagrams to evolve a software system. Since this often happens after the code evolution, it is difficult to remain in touch with the real changes in the code and it is easy to introduce a discrepancy between the system code and design information (Cazzola et al., 2005).

We propose to adapt code and design through the same mechanism (the maintenance rules), in this way no discrepancy will be introduced. Moreover, we also satisfy the urgency constraint because the adaptation is automatically performed.

2.2 Intermediate-Centric Layer

The intermediate-centric layer is the core component of the whole framework. It provides the system with the ability of manipulating its design information according to its evolution. It directly performs the manipulation on the XMI representation of the UML diagrams providing an API based on the logic concepts (diagrams, classes, relationships, and so on) and independent of the XMI syntax and complexity.

The intermediate centric layer has two benefits:

• it provides an abstract view of the design information that can be manipulated at run-time,
• it interfaces the data (design information) with the evolutionary application (the developer-centric layer) maintaining updated the data.

Moreover it provides a uniform approach to the design manipulation. Changing the design representation, the application does not change.

2.3 Developer-Centric Layer

The main role for the developer-centric layer is to keep the design information coherent with the evolved application. In that sense, the changes to the code must be reflected on the design information as well.

To achieve that, the developer implements the maintenance rules describing the designer point of view and how the application should evolve. The developer-centric layer is in charge of observing the application structure and behavior. If the application structure and/or behavior change, the layer detects these changes and applies the necessary maintenance rules reflecting the changes to the design information. The maintenance rules exploit the intermediate-centric layer to manipulate the design information. The developer must code the necessary maintenance rules when the new application behavior and structure is not captured by the available rules.

To really avoid the introduction of a discrepancy between code and design, the maintenance rules have to take care of adapting both the code and the design.

3 CASE STUDY: UTCS

The urban traffic control systems (UTCS) have a continuously changing nature. When designing the UTCS of a modern city, the software engineer must model both mobile entities (e.g., cars, pedestrians, vehicular flow, and so on) and fixed entities (e.g., roads, railways, level crossing, traffic lights and so on).

The software engineers, designing the UTCS, have to deal with a lot of unexpected and hard to plan problems of modern cities such as traffic lights disruptions, roads maintenance, car crashes, traffic jams, emergency routes and so on.

It is fairly evident the need for a self-adapting urban traffic control system capable of updating its design information as well. To this respect, we will
consider the area depicted in Fig. 1. It consists of two traffic nodes (tn_1 and tn_2); each traffic node represents a crossroads. The traffic flow at each traffic node is controlled by a set of traffic lights. In details, the traffic at the traffic nodes tn_1 and tn_2 are respectively controlled by four traffic lights. Both sets of traffic lights adopt the same synchronization protocol (named TwoGroupsSync): opposite traffic lights have always the same color, if a couple is red the other one is green or vice versa. The synchronization protocol specifies the following groups of synchronizations:

TwoGroupsSyn((A,B), [(tf_1, tf_3), (tf_2, tf_4)])
TwoGroupsSyn((A,B), [(tf_5, tf_7), (tf_6, tf_8)])

Note that, in the considered area we have a large avenue (the road composed by the sections rs_2, rs_4 and rs_7) with three lanes, the traffic lights steering the traffic flow in this avenue have three lights as well:

\[ \begin{align*}
  tf_2 &= \{tf_{2_1}, tf_{2_2}, tf_{2_3}\} \\
  tf_4 &= \{off, off, off\} \\
  tf_6 &= \{tf_{6_1}, tf_{6_2}, tf_{6_3}\} \\
  tf_8 &= \{off, off, off\}
\end{align*} \]

Figure 3(a) shows the statechart that describes the traffic lights behavior (synchronization policy) at the traffic node tn_1.

When an anomalous situation is detected (e.g., a traffic jam in the rush hour or a gas tube explodes) the UTCS must adapt itself to solve or alleviate the emergency. Of course, not all the anomalous situation can be foreseen at design-time and anyway the code and the design should not be polluted with the management of these anomalous and seldom cases. Therefore the adaptation dynamically takes place and consequently also the design must be changed.

Consider the case of the emergency plan, showed in Fig. 2, for alleviating the congestion at the rush hour in the large avenue. In the plan the first lane of the avenue will be run in the other direction and consequently some traffic lights change their behavior and the overall synchronization protocol.

In particular the traffic lights in the large avenue are characterized by:

\[ \begin{align*}
  tf_2 &= \{tf_{2_1}, tf_{2_2}, off\} \\
  tf_4 &= \{off, off, tf_{2_2}\} \\
  tf_6 &= \{tf_{6_1}, tf_{6_2}, on/off\} \\
  tf_8 &= \{off, off, off\}
\end{align*} \]

Figure 3(b) shows the statechart that describes the traffic lights behavior at the traffic node tn_1 after the application of the emergency plan.

3.1 The Role of the Developer

To realize the emergency plan (Fig. 2), the UTCS must evolve and its design information must be kept coherent with the performed evolution.

From the point of view of the developer, the design information must reflect the code changes necessary to realize the emergency plan and in particular he has to set how the design information has to change to be consistent to adapted system. To this regard, (s)he has to write the corresponding maintenance rules.
Figure 3: UML statecharts representing the traffic lights behavior at the traffic node tn1 before and after the adaptation.
3.1.1 Maintenance Plan

The maintenance rules will apply a set of operations to the design information that will transform them into the design information of the evolved system. In particular, in the considered example, the change to the traffic lights synchronization policy at the traffic node $tn_1$ implies to evolve the statechart in Fig. 3(a) to the statechart in Fig. 3(b) and to perform some adjustments to the deployment diagram.

Maintenance rules for the deployment model.

The maintenance rules has to carry out the following actions on the deployment diagram:

- turning on the traffic light $tf_4$ at the first lane and off at the second and third lanes;
- turning off the traffic light $tf_3$ at the third lane;
- creating a new synchronization protocol: TwoGroupsSyn((A,B), [(tf1, tf3), (tf4, tf5)]) for the changed set of traffic lights.

Maintenance rules for the statechart.

The maintenance rules has to carry out the following actions on the statechart:

- modifying as follows the $tf_2$ states: (1) to change the name of the composite state; (2) to add a new region; (3) to add a new composite state to the new region ($tf_{2_{state}}$ states with initial state=off);
- modifying as follows the $tf_4$ states: (1) to rename the composite state; (2) to turn on the composite state; (3) to add four states with required arcs and label (On, onOff, off, offOn); (4) to add a new region; (5) to add a composite state to the new region ($tf_{4_{state}}$ and $tf_{4_{state}}$) with initial state=off.

3.1.2 Maintenance Rules as Scripts

To automate the design information adaptation, the described rules must be implemented as scripts (e.g., Ruby or Python scripts) that can be invoked during the system evolution.

In the following we present some portions of the Ruby scripts necessary for adapting our test case. Once applied these scripts the design information will reflect the code adaptation and, for example, the statechart for the traffic lights synchronization will look as the statechart in Fig. 3(b).

This code snippet adapts the synchronization between traffic lights by adding a new region at the statechart with a state called "tf4−lane1 states".

```ruby
# add the transitions

top1SiTl4lane1r = top1SiTl4lane1r.addState("on")
top1SiTl4lane1rOnOff.addState("onOff")
top1SiTl4lane1roff = top1SiTl4lane1r.addState("off")
top1SiTl4lane1roffOn = top1SiTl4lane1r.addState("offOn")
top1SiTl4lane1roffOn.addTransitionTo (top1SiTl4lane1ronOff, ",
"t=30 sec", "tick()")
top1SiTl4lane1ronOnOff.addTransitionTo (top1SiTl4lane1roff, ",
"t=5 sec", "tick()")
top1SiTl4lane1roff.addTransitionTo (top1SiTl4lane1ron, ",
"t=20 sec", "tick()")
top1SiTl4lane1roffOn.addTransitionTo (top1SiTl4lane1ron, ",
"t=5 sec", "tick()")
```

4 RELATED WORK

Maintenance and evolution of continuously software systems is becoming an interesting topic of investigation. Here we relate our work to research on software evolution, UML refactoring, reflective and adaptive techniques to software evolution.

The methodology for defining the relation between the what (i.e., understanding) of software evolution and the how (i.e., control and support) of software evolution presented in (Lehman and Ramil, 2003). In (Lehman et al., 2002), the system dynamic model that aids to formalize the behavioral model of the development processes for the long-lived systems.

Refactoring techniques help to overcome the problems at the code-level by defining software transformations that restructure a software system while preserving its behavior. Proposal for refactoring of UML models presented in (Suný et al., 2001).

Maintaining the consistency between UML models has been presented in (Van Der Straeten et al., 2003).
In (Chiorean et al., 2004), the authors have presented a practical approach to check the consistency between UML models by using OCL based on transfer UML model by using the standard XMI. Whereas, a method for tracing the concurrent Java programs by using the UML is presented in (Mehner, 2002).

In (Cazzola et al., 2004), the authors have presented RAMSES, a reflective architecture that provides an application with the ability to self-adapt based on its design information. In this paper, we are describing a framework that performs the vice versa of the RAMSES middleware, by keeping the design information coherent with the self-adapting application.

In (Yoder and Johnson, 2002), the authors have presented an approach named adaptive object model that helps both architects and developers to understand, develop, and maintain systems. This approach provides an aspect-oriented model of the application that can change whenever a business change is needed and be immediately reflected on the running application.

The above approaches deal with adaptation and transformation models, similar solutions are required for adapting the design information of the software systems. We consider our approach as a method, that supports the online evolution for the design models of the software systems.

5 CONCLUSION

In this paper we have illustrated how to maintain the design information of the software system based on their internal representation stored in XMI schema. The approach permits of evolving the design information consistently with the evolution of the application. We have shown the applicability of the approach on a case study.

The benefit for the proposed approach, is to save the time and efforts and increase the performance. The developer implements the changes in form of script rules, and apply them to the XMI schema when the system is evolved.

ACKNOWLEDGEMENTS

The RAMSES project is funded by the Deutsche Forschungsgesellschaft (German Science Foundation), project number SA 465/31-1.

REFERENCES


Keywords: Web-based Applications, SysML, UML 2.0, Modeling language.

Abstract: This paper discusses the importance of a new modelling language SysML (system modelling language) and shows how it differs from UML2.0 (unified modelling language) in the development of web-based applications. The development of Web applications has become more complex and challenging than most of us think. In many ways, it is also different and more complex than traditional software development and there is a lack of a proven methodology that guides software engineers in building web-based applications. In this paper we recommended using SysML for building and designing web-based applications.

1 INTRODUCTION

In the span of a decade, the World Wide Web has become ubiquitous, and it continues to grow unabated at exponential rate. Web-based systems and applications now deliver a complex array of varied content and functionality to a large number of heterogeneous users. The interaction between a Web system and its backend information systems has also become more tight and complex (San Murugesan, 2001).

Although the development of web-based applications made many improvements, there is still a lack of an established software engineering methodology for constructing web-based systems. Consequently, much of the development is carried out without a true understanding of analysis and design issues. Currently, the problems of developing web-based systems are similar to those in traditional software engineering thirty years ago where programming and performance were the main issues. However, just as the focus in traditional software focus with web-based systems must shift from technical issues to the development process (Athula Ginige, 2002).

UML-based Web Engineering (UWE) is a development process for Web applications with focus on systematic design, personalization and semi-automatic generation. UWE describes a systematic design methodology using exclusively UML (Unified Modelling Language) techniques, the UML notation and the UML extension mechanisms (Nora Koch, 2002).

The Systems Modelling Language (SysML) is a new visual language designed by systems engineers. SysML supports the specification, analysis, design, verification and validation of a broad range of systems. These may include hardware, software, information, processes, personnel, and facilities. SysML extends UML 2.0 with additional constructs appropriate for complete systems modelling (SysML Partners, 2003).

2 COMPONENTS OF WEB-BASED SYSTEMS

Web-based systems rely on three-tier architecture: The client, the web server and the database:

The client. The client provides the user interface for the web-based application. It is of crucial importance, since web applications are more user-oriented than traditional systems. We can use web tools to automate some of the tasks of designing the web interface by generating the HTML code, e.g.
manipulating tables, colors, and other web elements. However, the construction of web-based user interfaces must rely on principles rooted in human-computer interaction.

The web server. The web server provides the business logic of web applications. It is responsible for interacting with the client and the database. The web server accepts a user request for data from the client, retrieves the data from the database, and then responds to the client request. We use Java servlets and CGI scripts for implementing the web server. In practice, a Java servlet works in much the same as a CGI script. The difference between them is the ease of use.

The database. The database maintains the data needed for the web application. The web server can communicate with the database via JDBC (Java Database Connectivity).

The JDBC is built around the Structured Query Language (SQL) which can be used to manipulate a variety of databases without having to deal with the specificity of those databases (Said Hadjerrouit, 2001).

3 UML 2.0

The Unified Modeling Language (UML) was launched in 1995 and adopted as an industry standard by the Object Management Group (OMG) in 1997 (OMG 2002). Since then, its use has been steadily increasing in both industry and academia to the point where it has become the prevalent general-purpose modeling language. As experience with UML grew and the issues and needs of software modeling became better understood, new requirements for UML emerged. This led to the issuing of formal requests for the first major revision of the standard. The requirements called for increased precision, greater clarity of the specification, and some new modeling capabilities.

Concurrently with the publication of the requirements for UML 2.0, and inspired in a large part by the widespread adoption of UML, the OMG launched its Model-Driven Architecture (MDA) initiative. This defines a conceptual framework for a model-driven approach to software development and, based on that, a roadmap for a set of corresponding industry standards.

This had a significant impact on the ultimate form of UML2.0, since one of the key elements of MDA is the potential for using modelling languages for more than just documentation and high-level design “sketching”. This includes the abilities to automatically generate implementations from models or to perform complex formal analyses to determine the soundness and validity of proposed designs. In fact, supporting automation is one of the cornerstones of MDA. This means the use of computers to mechanize some of the more complex repetitive activities involved in software development that were traditionally by programmers. Needless to say, automation is one of the most effective technological means for improving productivity and product reliability.

3.1 Overview of UML-based Web Engineering Developing Process

The developing process consists of four steps. These steps are the requirements analysis, conceptual, navigation and presentation design. They produce the following artifacts:

- use case model
- conceptual model
- navigation space model
- navigation structure model
- presentation model

These models are refined in successive iterations of the UWE development process. Figure 1 shows the models.

Represented as UML packages related by trace dependencies (process relationship).

The goal of the requirements analysis is to find the functional requirements of the Web application and to represent these requirements as use cases.

The objective of the conceptual design is to build a conceptual model of the application domain taking into account the requirements captured with use cases. Traditional object-oriented techniques are used to construct the conceptual model, such as finding classes and associations and defining inheritance structures. The conceptual model is represented by an ordinary UML class diagram.

Based on the conceptual model the navigation method proposes a set of guidelines to construct a navigation model which represents the navigation space and the navigation structure by adding access elements that can be used for navigation. The method includes a set of UML stereotyped modelling elements for navigation design, like indexes, guided tours, queries and menus.

These stereotypes are used in the construction of UML class diagrams to represent the navigation space model and the navigation structure model. Presentation modelling aims at the design of abstract user interfaces and the design of the user interaction with the Web application. It consists of two steps: The first step in the presentation design defines user interface views which sketch the content and the look and feel of the nodes. These user interface views can then be combined to storyboarding
scenarios. The second step focuses on the dynamics of the presentation represented with UML sequence diagrams (Jacobson I, Booch, 99).

3.2 The Method

They apply the steps suggested by many use case driven processes (Kruchten, 99) to build the use case model of a Web application. These steps are:
1. Find the actors.
2. For each actor search the text for activities the actor will perform.
3. Group activities to use cases.
4. Establish relationships between actors and use cases.
5. Establish “include” and “extends” relationships between use cases.
6. Simplify the use case model by defining inheritance relationships between actors and/or between use cases.

For each use case a detailed description can be provided in terms of (primary and secondary) scenarios, for instance following the guidelines of Schneider and winters (Schneider G, 98). The activities flow of activities related to a use case can be represented by a UML activity diagram.

3.3 Shortcomings of UML

Those who know UML, find it to be an effective modeling language. The roots of UML are firmly in software. OMG (Object Management Group, 2003) states that the “Unified Modeling Language (UML) is a general-purpose visual modeling language that is designed to specify, visualize, construct and document the artifacts of a software system.” However, many Systems Engineers believed the UML to be sufficiently flexible and robust to support extensions to address the needs of systems engineering. One of the strengths of UML is its built-in mechanisms for specializing the generic forms of its modeling elements to more application-specific variants. Collectively, these provide a capability for UML “Profiles” that package specific terminology and substructures for a particular application domain. Exploiting this had the potential to achieve a “standard modelling language for Systems Engineering to analyze, specify, design, and verify complex systems, intended to enhance system quality, improve the ability to exchange Systems Engineering information amongst tools, and help bridge the semantic gap between systems, software, and other engineering disciplines” (SysML Object Management Group, 2003). However, the modifications to UML needed for Systems Engineers require more than just the addition of stereotypes.

3.4 Problems with UML 2.0

UML 2.0 went some way towards addressing the problems of modelling architectures. The Structured Class Diagram provides a hierarchical architecture; however, it only allows one level of hierarchy and does not allow the modelling of flows on links. Links to requirements, parametric equations, and others were also not addressed.

4 SYSML

SysML supports the specification, analysis, design, verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities.

The origins of the SysML initiative can be traced to a strategic decision by the International Council on Systems Engineering’s (INCOSE) Model Driven Systems Design workgroup in January 2001 to customize the Unified Modeling Language (UML) for systems engineering applications. This resulted in a collaborative effort between INCOSE and the Object Management Group (OMG), which maintains the UML specification, to jointly charter the OMG Systems Engineering Domain Special Interest Group (SE DSIG) in July 2001. The SE DSIG, with support from INCOSE and the ISO AP 233 workgroup, developed the requirements for the modeling language, which were subsequently issued by the OMG as part of the UML for Systems Engineering Request for Proposal in March 2003.

Currently it is common practice for systems engineers to use a wide range of modelling languages, tools and techniques on large systems projects (SysML Partners, 2003)

4.1 Compliance

As with UML, the basic units of compliance for SysML are the packages which define the SysML met model.

There are two kinds of SysML compliance. The first kind of compliance is concerned with defining the subset of UML 2 Superstructure (UML) packages required to implement SysML. The second kind of compliance is concerned with specifying the extent to which a SysML tool implements the SysML extensions to UML Superstructure.
In order to visualize the relationship between the UML and SysML languages, consider the diagram shown in Figure 2.

![SysML Diagram Taxonomy](www.sysml.org)

**Figure 2: SysML Diagram Taxonomy, adapted from www.sysml.org.**

5 WHY SYSML?

The main question in this paper is, “why should web-based application build and design be based on SysML?” The Unified Modeling language (UML) has, since its adoption in 1997, proved immensely popular with software engineers to the point where it is now the only widely used visual modeling language for software engineering. In March 2003, the OMG issued a Request for Proposal (RFP) for a customized version of UML suitable for Systems Engineering written by the SE DSIG (Object Management Group 2003) The customization of UML for systems engineering is intended to support modeling of a broad range of systems, which may include hardware, software, data, personnel, procedures, and facilities. There was only one technology submission to the RFP, which was by the SysML group, proposing a Systems Modeling Language, SysML.

OMG (Object Management Group, 2005) states “SysML supports the specification, analysis, design, verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities.” Equally, INCOSE (INCOSE, 2005) states that systems engineering is an “interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: Operations, Performance, Test, Manufacturing, Cost and Schedule, Training and Support, Disposal”.

**REFERENCES**

Athula Ginige , Proceedings of the 14th international conference on Software engineering and knowledge engineering SEKE ’02 July 2002


REVERSE ENGINEERING ELECTRONIC SERVICES

From e-Forms to Knowledge

Costas Vassilakis, George Lepouras, Akrivi Katifori
Department of Computer Science and Technology, University of Peloponnese, Karaiskaki 22100, Tripoli, Greece
costas@uop.gr, gl@uop.gr, katifori@uop.gr

Keywords: e-Government, electronic services, reverse engineering, organizational knowledge.

Abstract: On their route to e-governance, public administrations have developed e-services. Each e-service encompasses a significant amount of knowledge in the form of examples, help texts, legislation excerpts, validation checks etc. This knowledge has been offered by domain experts in the phases of service analysis, design and implementation, being however bundled within the software, it cannot be readily retrieved and used in other organizational processes, including the development of new services. In this paper, we present an approach for reverse engineering e-services, in order to formulate knowledge items of a high level of abstraction, which can be made available to the employees of the organizations. Moreover, the knowledge items formulated in the reverse engineering process are stored into a knowledge-based e-service development platform, making them readily available for use in the development of other services.

1 INTRODUCTION

In the past few years, governments are realizing e-government policies and frameworks, which include delivery of e-services for enterprises and citizens. In this context, development of an e-service is usually treated as an isolated project, thus information extracted from domain experts in the analysis phase is recorded as low level “user requirements”, rather than as high-level knowledge (Vassilakis, 2003). This practice leads to suboptimal results since:

• the “software specifications” format is inappropriate for knowledge sharing among the organization’s employees. Employee groups that could benefit from the knowledge amassed during the analysis phase include domain experts, seeking information on relevant subjects and help desk workers, who could use this knowledge to provide information and guidance to users of the e-service.

• the knowledge offered by domain experts, includes a number of examples, explanations, related legislation and so forth; in this form, it could be used to tackle the “lack of expert assistance” usage barrier for e-services identified in (Vassilakis, 2005), according to which users refrain from using e-services because no adequate help is available.

• software specifications produced for an e-service are usually considered as pertinent to the specific service only; this reduces opportunities for reusing the knowledge for developing other services (e.g. re-using the personal details portion of a form).

To tackle these deficiencies, organizations are adopting either (a) knowledge management (KM) platforms, for recording knowledge in explicit format and facilitate searching, browsing and sharing and (b) e-service development platforms, which can leverage component reusability across services. For already developed services, however, the original knowledge has already been mapped to software specifications and artifacts (HTML forms, JavaScript/back-end code, database schemata etc), therefore these services must either remain “isolated islands” or be remodeled in the chosen platform (KM or e-service development platform), incurring thus additional effort and cost.

In this paper we present a method for reverse engineering software components of developed e-services, and using the individual elements identified in the reverse engineering process to synthesize artifacts of higher levels of abstraction. These artifacts encompass aspects useful both for KM and e-service development, being consequently suitable both for knowledge sharing and dissemination within the organization, as well as for developing new services. The presented method has been applied to produce artifacts suitable for importing into the SmartGov platform, a knowledge-based development environment for public sector online services (Georgiadis, 2002), (SmartGov, 2004).
The rest of the paper is organized as follows: section 2 presents related work. Section 3 introduces the SmartGov platform, while section 4 elaborates on the reverse engineering process. Finally, section 5 concludes the paper and outlines future work.

2 RELATED WORK

Reverse engineering is a process of examination (as opposed to alteration), directly supporting the essence of program understanding: identifying artifacts, discovering relationships, and generating abstractions (Chikofsky, 1990). Reverse engineering methods and techniques are used for three canonical activities, namely data gathering, knowledge management and information exploration (Tilley, 2000). The activity of knowledge management in particular, refers to capturing, organizing, understanding, and extending past experiences, processes, and individual know-how. In this context, the reverse engineering process produces artifacts that, if properly managed, could be shared at various levels, e.g. development team or department, serving thus as an active repository of corporate knowledge (Kazman, 1998). Regarding the application of software reverse engineering techniques on web applications, notable activities reported insofar include (DiLucca, 2004), which aims to the construction of UML diagrams so as to support the maintenance and evolution of web applications; (Paganelli, 2003) describes a method for extracting task models from web pages, in order to reconstruct the underlying interaction design; finally RetroWeb (Essanaa, 2004) aims at providing a description of the informative content of the site at various abstraction levels: physical, logical and conceptual.

3 THE SMARTGOV PLATFORM

The SmartGov platform offers functionality for managing knowledge and validation rules, creating objects, designing forms and services and deploying them. The central concept in the SmartGov platform is that of Transaction Service Elements (TSEs). TSEs are in fact widgets, which can be used as building blocks for e-services. Contrary though to user interface widgets, TSEs extend beyond visual appearance: they can contain metadata and domain knowledge. Metadata include the object’s type, labels, allowable values, validation checks, and online help, while domain knowledge includes relationships to other elements, documentation, legislation information etc. Other concepts in the platform are TSE groups (assemblies of individual TSEs which can be managed collectively), forms (canvases on which TSEs and TSE groups are placed) and transaction services (TSs – collections of forms offering a specific service). Similarly to TSEs, instances of these concepts contain metadata and domain knowledge. Metadata elements in these concepts vary according to the concept type, e.g. metadata for a TS include the authentication method and whether modification of submitted documents is allowed. The SmartGov platform also offers functionalities for establishing links among instances of the modeling concepts (TSEs, groups, forms and TSs), formulating thus a semantically rich network of elements, which can be browsed or queried by platform users. For more information on the SmartGov platform, refer to (SmartGov, 2004).

4 REVERSE ENGINEERING E-SERVICES

In this section we present the rules employed for electronic artifact identification and composition. The aim of this reverse engineering approach is to formulate semantically rich artifacts (TSEs, TSE groups, forms and TSs), with each one of them encompassing visual characteristics, knowledge (help texts, examples, etc), business rules (validation checks) and relations with other elements. The heuristics for combining individual HTML form elements into e-service artifacts exploit the structure and nesting of HTML tags, naming conventions and element proximity. In the rest of this section, we first present the common artifact patterns as they appear in transactional services; these patterns dictate the operation of the heuristics for artifact identification and composition, which is discussed subsequently.

4.1 Artifact Patterns in e-Services

When designers create the pages comprising a service, they arrange elements in ways that are meaningful and usable for service users. In order to extract patterns for these arrangements, more than 50 online services from different countries were examined. The services used in this analysis were selected from well-established government portals for online services, including the US portal (www.firstgov.gov), the UK online service directory (www.direct.gov.uk) and the Singapore e-government services for citizens catalogue (www.ecitizen.gov.sg). The layout of printed forms
A common layout for service elements is shown in fig. 1, where we can identify the following areas:

1. a **header**, including the agency logo, links to the agency’s home page and generic help, as well as a graphic acting as a separator,
2. the **main body**, which includes short introduction of the form, the actual input elements (grouped here in four areas) and their explanations,
3. and a **footer**, including navigational controls (Continue button) and a service-specific help link.

Input elements have been organized into groups, with each group having a header (e.g. Name, OLD address). A help hyperlink (for the “Name” group) and/or some text for the group as a whole (e.g. in the “Email Address” group) may also be present. Within a group, input elements may be laid out either (a) horizontally, with their descriptions being placed above (below) them (e.g. the “Name” group) or (b) vertically, with their descriptions being placed on the left of the element (more rarely, on the right).

In both layouts, help texts and additional help or utility links for individual input elements may be present, which may be placed besides the field description or the input element. Examples of help texts and additional help links are illustrated in fig. 2, while an example of a utility link is the “Zip Code lookup”, next to the zip code inputs in fig. 1. In some cases, a particular data item may be collected using more than one input element, as is the case of the date of birth (fig. 2) and the “SSN” data item (fig. 3). Typically, this technique is used for registration numbers (SSN, bank account numbers, license plates etc), as well as dates. In such cases, the constituent input elements are usually juxtaposed on the layout, with the possible intervening of a separator (dash, slash, space and so forth). Notice that the overall form layouts in figs. 2 and 3 follow the pattern identified for fig. 1. An additional commonplace practice is the use of the asterisk (*) to denote mandatory fields (figs. 2 and 3). The asterisk is most usually placed next to the input area or next to the label.

### 4.2 Artifact Identification & Creation

The phase of artifact identification begins with the specification of the HTML pages that comprise the service. The pages may be read directly from the web server hosting the service, or from a local file system. This phase includes application of heuristics that attempt to recognize the patterns described in the previous section within the HTML pages. For each pattern identified, a proper artifact is constructed, encompassing all information pertinent to it; if appropriate, links to other artifacts are also established. Tag nesting, JavaScript code associated with HTML page elements and naming conventions are additional sources of information for the reverse engineering process.
Before the application of heuristics, the reverse engineering software (RES) creates the object model of each page, i.e. a tree-structured representation of the page components (tables, divisions, forms, fields etc. The HTMLParser (htmlparser.sourceforge.net) package was chosen for this purpose. The heuristics for each type of component (TSE, TSE group, form, and TS) are presented in the following paragraphs.

### 4.2.1 Identifying Transaction Service Elements

A transaction service element (TSE) in the SmartGov platform is a compound object encompassing the input area and its properties (HTML input type, size, maximum length, initial value), the input area label, help texts (commonly provided as hyperlinks or as extended in-place text), the validation rules that apply to the values entered (data type, mandatory input, allowable ranges etc) and, finally, its relationships with other elements.

The first task towards TSEs identification is to locate the widgets allowing for data input. HTML provides four basic input widgets, namely *input, select, textarea* and *button*. For each such construct a respective TSE is created, except for the case of inputs of type *radio*, for which a single TSE is created for all input instances with the same value for the *name* attribute. The reverse engineering process subsequently locates information for the additional aspects of the TSE as follows:

Firstly, the TSE label is determined. The form is initially scanned for a *label* element whose *for* tag matches the input element name (e.g. `<label for="fname">First Name</label>`), or for a *label* element enclosing the input area definition (e.g. `<label>First Name <input type="text" name="fname" /></label>`). If such an element is found, the text specified in the *label* element is used as the TSE label. If no such label is found, the RES attempts to determine the label by its positioning relative to the input area: the label may be placed on the left of the input area (figs. 2, 3 and bottom half of fig. 1), or above the input area (upper half of fig. 1). Note that the text may be formatted using tables, thus “left” does not necessarily refer to HTML code immediately preceding the input tag, but may be the text included in the table cell appearing on the left of the field under examination. The RES takes into account the case that an extra column, indicating whether the field is mandatory or not, intervenes between the input area and the label field (fig. 2).

Afterwards, the help items for the field are located. The help items may be located at the right of the input area, either as directly following HTML code (fig. 1) or within an adjacent table cell (fig. 2). In some cases, only a hyperlink may be present which has to be clicked to display the help content. In such cases, the RES retrieves the content pointed to by the help anchor, and packs this content within the TSE; the label text (determined in the previous step) is also scanned for presence of hyperlinks. If such hyperlinks are found, the content pointed to by each hyperlink is extracted and packed with the TSE as a help item. This step may produce multiple help items for a single TSE. Additional help items may be determined from code analysis (described below).

The next step is to extract an initial indication whether a TSE is considered mandatory or not. The presence of an asterisk either packed within the label (at its beginning or end – fig. 3) or as a separate table column (fig. 2) is used as such an initial indication. An additional check to determine whether some input element is mandatory or not is performed in the code analysis phase (see below).

Subsequently, the default value for the input area is determined by examining the settings of the HTML attributes associated with the input area (e.g. the “value” attribute for text boxes and buttons, the “checked” attribute for check boxes etc). The values of the “maxlength”, “size”, “rows” and “cols” attributes, whenever present, are also extracted and bundled as properties of the TSE under construction.

For input elements with a closed set of values (such as select widgets and radio buttons), the set of values is examined to determine the data type of the input element. If all the values within the set are of the same type (integers, floats, dates, etc), the data type of the TSE under construction is set accordingly; otherwise, the data type is set to “string”. Data type inference for input elements with an open set of values (free user type-in) is handled through code analysis (described below).

The TSE properties listed above can be directly determined form attributes values of the input elements or from text placement in relation to the input element. However, some important aspects of TSEs, namely the data type, whether a TSE is read-only or not, as well as validation checks may not be directly modeled as attribute values; instead, e-service developers use JavaScript to provide these features. In order to determine these features, the RES analyzes the JavaScript code associated with input element events. This analysis may also reveal additional help items and supplementary indications on whether the TSE is mandatory or not. JavaScript code analysis is based on heuristics, since rigorous semantic analysis was considered exaggerate for the issues at hand, taking also into account that the
results will be reviewed by humans before being used for code generation. These heuristics are:

1. if the “onFocus” and “onSelect” event handlers of the input element are present and contain code that moves the focus away from the field (typically this is performed using the `this.blur()` method or by moving the focus to another field through the `anotherfield.focus()` method), then the TSE is characterized as “read-only”. Note that this is complementary to checking for existence of the “readonly” and “disabled” input element attributes, i.e. if either of the checks succeeds, the TSE is characterized as “read-only”.

2. if the JavaScript code within the page contains instructions that compare the value of the element with the empty string (`elem.value == ''` or `elem.value.length == 0`) and emit a message if the condition is true, then the TSE is considered mandatory. Code patterns that trim the spaces from the element value and compare the result with the empty string (e.g. `trim(elem.value) == ''`) are also taken into account in this check.

3. if the “onFocus”, “onSelect” and “onMouseOver” event handlers of the input element exist and contain code displaying text on the browser status bar (e.g. `onfocus="javascript:window.status = 'Enter net income'"`) or at some other page element (e.g. `onfocus= "javascript:document.getElementById('helpArea').innerHTML="Enter net income""`) then the displayed text is considered an extra help item for the TSE under construction.

4. if the “onChange” event handler exists, then the code in it is scanned for function invocations whose argument list does not reference other fields. The name of each such function is examined to determine whether it is a compound word, whose first component is one of the words “check”, “is”, “valid”, “validate”, “verify”, while the second component being a data type name or a synonym for it (number, date, integer, float, numeric and so forth) –e.g. `onChange="checkNumber(this, 'Price should be a number');"`. If a match is found, the data type for the TSE under construction is set accordingly. The whole JavaScript code of the page is also scanned for conditions of the form `if (checkNumber(price)...)`, to cater for cases that user input validation is deferred until form submittal, rather than being performed synchronously with data typing.

5. code associated with the “onChange” event handler and that (a) does not reference other fields (b) does not meet the naming criteria of item (4) and (c) emitting a message, is recorded as a validation check for the TSE. This code may implement any validation check e.g. value range, data format and so on. Conditions of `if` statements anywhere within the JavaScript code of the page that reference only the specific TSE are added -together with the associated code block- to the list of validation checks associated with the TSE.

At this stage, all data regarding the TSE artifact have been collected, and the TSE is finalized.

### 4.2.2 Identifying TSE Groups

The HTML standard provides the `fieldset` tag for specifying groups of fields. Browsers supporting this feature draw a border surrounding the input areas (fig. 4) to provide a visual clue that these elements are logically associated. The field set may be assigned a label using a nested `legend` tag. The RES identifies such constructs and for each one of them creates a TSE group artifact, which is automatically linked with the individual TSEs it contains. The TSE group description is derived from the contents of the enclosed `label` element, while extra text occurring within the `fieldset` construct and not directly associated with a specific TSE (e.g. the `Please enter...` phrase in fig. 4) is considered as a detailed description for the TSE group. Hyperlinks occurring within such extra text are considered as help items for the TSE group as a whole. The TSE group under construction is finalized by adding to it the pertinent validation checks. These are identified as follows:

- the `onChange` event handler of the TSE elements belonging to the group are scanned for code that involves two or more elements of the group (e.g. `onChange="check_date(day, month, year)"`) but not referencing any field outside the group.
- the page’s JavaScript is scanned for `if` statements’ conditions involving two or more members of the TSE group, but not referencing any field outside the group. These conditions, together with the associated action blocks, are added to the list of validation checks associated with the TSE group.

At this stage, all data regarding the TSE group have been collected, and the artifact is finalized.

The `fieldset` tag is not however the predominant approach for implementing field groups: tables are usually employed instead since (a) not all browsers support the `fieldset` tag and (b) tables provide more flexibility for laying out titles, borders, fields etc. The RES deduces field groups by identifying `table`
segments: a table segment comprises of a header row containing only text (cf. fig. 1, rows “OLD Address”, “NEW Address”, “Email address” and Fig. 2, row “Type of passport”), followed by a number of body rows, containing labels, input widgets and help texts. For each table segment, a TSE group is created; the text within the header row is used as the TSE group description, and links to the TSEs corresponding to the input fields within the text segment are established. Processing for help items and validation checks proceeds as described for TSE groups defined using the fieldset tag.

4.2.3 Creating Form Artifacts

For each file processed, the RES produces one form artifact. The form artifact is linked to the TSE and TSE group artifacts it contains, while the form header and form footer areas (i.e. HTML code before the first TSE/TSE group and HTML code after the last TSE/TSE group respectively) are used to populate the respective elements of the form artifact. Hyperlinks within the form header and footer are exploited to create help items for the form, as previously described for TSEs. Validation checks involving multiple fields not belonging to the same TSE group are finally added to the form artifact, as validation checks pertaining to the form as a whole.

4.2.4 Creating the Transaction Service Artifact

For each invocation, the RES constructs a single TS artifact. This contains links to the service forms, and each such link is tagged with the order that the form appears in the service. Once the TS artifact has been formulated, all artifacts are imported into the SmartGov platform, made thus available for use in developing other services. The reverse engineered service itself may be re-generated, by invoking the SmartGov platform’s Integrator module.

5 CONCLUSIONS

In this paper we have presented a method for reverse-engineering e-services into artifacts of higher level of abstraction, which may be used for knowledge representation and sharing, and as reusable components for development of other services. Future work will include co-examination of the back-end code (e.g. PHP, JSP), to reveal more validation checks, handling of multilingual service aspects and generalization of “quite similar” artifacts for the creation of more generic artifact templates.

ACKNOWLEDGEMENTS

This work has been partially funded by the “Intelligent Historical Archive Document Management”/PENED 2003 project.

REFERENCES

A SCENARIO GENERATION METHOD USING A DIFFERENTIAL SCENARIO

Masayuki Makino, Atsushi Ohnishi
Department of Computer Science, Ristumeikan University, Kusatsu, Shiga 525-8577, Japan
makino61@selab.cs.ritsumei.ac.jp, ohnishi@cs.ritsumei.ac.jp

Keywords: Scenario analysis, Scenario generation, Requirements elicitation, Requirements definition.

Abstract: A generation method of scenarios using differential information between normal scenarios is presented. Behaviours of normal scenarios belonging to the same problem domain are quite similar. We derive the differential information between them and apply the information to generate new scenarios. Our method will be illustrated with an example.

1 INTRODUCTION

Scenarios are important in software development, particularly in requirements engineering, by providing concrete system description (Weidenhaupt et al., 1998). Especially, scenarios are useful in defining system behaviors by system developers and validating the requirements by customers. In scenario-based software development, incorrect scenarios will have a negative impact on the overall system development process. However scenarios are usually informal and it is difficult to verify the correctness of scenarios.

The authors have developed a scenario language for describing scenarios in which simple action traces are embellished to include typed frames based on a simple case grammar of actions and for describing the sequence among events (Zhang et al., 2004). Since this language is a controlled language, the vagueness of the scenario written with this language can be reduced. Furthermore, the scenario with this language can be transformed into internal representation. In the transformation, both the lack of cases and the illegal usage of noun types can be detected (Ohnishi, 1996).

Scenarios can be classified into 1) normal scenario, 2) alternative scenario, and 3) exceptional scenario. A normal scenario represents the normal and typical behavior of the target system, while an alternative scenario represents normal but untypical behavior of the system and an exceptional scenario represents abnormal behavior of the system. In order to grasp whole behaviors of the system, not only normal scenarios, but also alternative/exceptional scenarios should be specified. However it is difficult to hit upon most of alternative scenarios and exceptional scenarios, whereas it is easy to think of normal scenarios.

2 SCENARIO LANGUAGE

2.1 Outline

The scenario language named SLAF has already been introduced (Zhang, 2004, Toyama 2005). In this paper, a brief description of this language will be given for convenience.

A scenario can be regarded as a sequence of events. Events are behaviors employed by users or the system for accomplishing their goals. We assume that each event has just one verb, and that each verb has its own case structure (Fillmore, 1968). The scenario language has been developed based on this concept. Verbs and their own case structures depend on problem domains, but the roles of cases are independent of problem domains. The roles include agent, object, recipient, instrument, source, etc.

We provide requirements frames (Ohnishi, 1996) in which verbs and their own case structures are specified. The requirements frame depends on problem domains. Each action has its case structure, and each event can be automatically transformed into internal representation based on the frame. In the transformation, concrete words will be assigned to pronouns and omitted indispensable cases. With Requirements Frame, we can detect both the lack of cases and the illegal usage of noun types.

We assume four kinds of time sequences among
events: 1) sequential, 2) selective, 3) iterative, and 4) parallel. Actually most events are sequential events.

2.2 Scenario Example

We consider a scenario of train ticket reservation of a railway company. Figure 1 shows a scenario of customer’s purchasing a ticket of express train at a service center of a railway company. This scenario is written with our scenario language based on videoized behaviors of both a user and a staff at a service center of a railway company (Railway Information System, 2001).

A title of the scenario is given at the first line of the scenario in Fig.1. Viewpoints of the scenario are specified at the third line. In this paper, viewpoints mean active objects such as human and system appearing in the scenario. There exist two viewpoints, namely staff and customer. The order of the specified viewpoints means the priority.

In this scenario, almost all events are sequential, except for just two selective events (the 9th event and the 13th event). Selection can be expressed with if-then syntax like program languages. Actually, event number is for reader’s convenience and not necessary.

2.3 Analysis of Events

Each of events is automatically transformed into internal representation. For example, the 2nd event “The staff sends the customer’s request to reservation center via private line” can be transformed into internal representation shown in Table 1.

In this event, the verb “send” corresponds to the concept “data flow.” The data flow concept has its own case structure with four cases, namely to say, source case, goal case, object case and instrument case. Sender corresponds to the source case and receiver corresponds to the goal case. Data transferred from source case to goal case corresponds to the object case. Device for sending data corresponds to the instrument case. In this event, “customer’s request” corresponds to the object case and “the staff” corresponds to the source case.

The internal representation is independent of surface representation of the event. Suppose other representations of event, “Customer’s request is sent from staff to reservation center via private line” and “reservation center receives customer’s request from staff via private line.” These events are syntactically different but semantically same as the 2nd event. These two events can be automatically transformed into the same internal representations.

3 DIFFERENTIAL SCENARIO

Systems belonging to the same domain similarly behave each other. In other words, normal scenarios belonging to the same domain resemble each other. Since our scenario language provides limited vocabulary and limited grammar, the abstraction level of any scenarios becomes almost same.

For one system, there exist several normal scenarios. In case of ticket reservation, reservation can be written as a normal scenario and cancellation can be written as another normal scenario. To make a differential scenario, we select two normal scenarios of two different systems. Each of the two scenarios represents almost same behavior, such as reservation of a ticket.

The differential scenario consists of 1) a list of corresponding words, 2) deleted events which appear in one scenario (say, scenario A) and do not appear in the other (say, scenario B), and 3) added events which do not appear in scenario A and appear in scenario B.

Fig. 2 shows a scenario of flight ticket reservation using credit card. By comparing two scenarios, we can get the differential scenario. The first four events of the scenario in Fig. 1 can be transformed as shown in Table 2. In fact, data flow...
concept has four cases, that is, source, goal, object, and instrument cases as shown in Table 1, but the instrument cases are omitted in Table 2 and 3 for the space limitation.

Since the sequence of the concepts of the first four events of the scenario in Fig. 1 is same as that of the scenario in Fig. 2, we can regard these events are corresponding each other. Then, the difference between cases of the corresponding events will be checked. In the case of the first event of the two scenarios, object cases of the events are different each other.

Table 2: The internal representation of the first four events of the scenario in Fig. 1.

<table>
<thead>
<tr>
<th>concept</th>
<th>agent/source</th>
<th>goal</th>
<th>objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>staff</td>
<td>customer</td>
<td>leaving station, destination,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>traveling date</td>
</tr>
<tr>
<td>data flow</td>
<td>staff</td>
<td>reservation center</td>
<td>customer's request</td>
</tr>
<tr>
<td>retrieve</td>
<td>staff</td>
<td>available trains</td>
<td>request</td>
</tr>
<tr>
<td>data flow</td>
<td>staff</td>
<td>customer</td>
<td>list of available trains</td>
</tr>
</tbody>
</table>

Table 3: The internal representation of the first four events of the scenario in Fig. 2.

<table>
<thead>
<tr>
<th>concept</th>
<th>agent/source</th>
<th>goal</th>
<th>objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>staff</td>
<td>customer</td>
<td>leaving airport, destination,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>traveling date</td>
</tr>
<tr>
<td>data flow</td>
<td>staff</td>
<td>reservation center</td>
<td>customer's request</td>
</tr>
<tr>
<td>retrieve</td>
<td>staff</td>
<td>available flights</td>
<td>request</td>
</tr>
<tr>
<td>data flow</td>
<td>staff</td>
<td>customer</td>
<td>list of available flights</td>
</tr>
</tbody>
</table>

The difference between corresponding events will be stored as corresponding words in Table 4. The 12th and the 13th events of Fig. 2 are not-corresponding events and will be stored as added events, while the 12th event of Fig. 1 and the 14th event of Fig. 2 are corresponding events. The 13th event of Fig. 1 is a not-corresponding event and will be stored as a deleted event.

Finally, we can get the differential scenario between train ticket reservation and flight ticket reservation shown in Table 4, 5, and 6.

Table 4: A list of corresponding words between scenarios of Figure 1 and 2.

<table>
<thead>
<tr>
<th>Fig.1</th>
<th>Fig.2</th>
<th>Fig.1</th>
<th>Fig.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>station</td>
<td>airport</td>
<td>trains</td>
<td>flights</td>
</tr>
<tr>
<td>traveling</td>
<td>departure</td>
<td>cash</td>
<td>credit card</td>
</tr>
<tr>
<td>train</td>
<td>flight</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Added events.

The staff checks the credit card.
The staff charges the ticket fee to the card.

Table 6: Deleted events.

If (changes exist) then the staff gives changes.

4 SCENARIO GENERATION

Once differential scenario between system A and B given, we can apply it to another scenario of system A and get a new scenario of system B by changing corresponding words and by deleting or adding not-corresponding events.

Fig. 3 shows an exceptional scenario of ticket reservation. In this scenario, the customer cannot get any available trains with respect to the first request. So, the customer changes the traveling date and then gets available trains.

By applying the differential scenario in Table 4, 5, and 6, we can get a new exceptional scenario of flight ticket reservation as shown in Fig. 4.

Figure 3: An exceptional scenario.
6 CONCLUSION

We have developed a generation method of scenarios using a differential scenario. Because of the space limitation, we showed just one example, but we confirmed that alternative scenarios and different normal scenarios can be generated with our method.

We have to validate the ideas more thoroughly by applying to several different problem domains. We have been developing a prototype system based on the method. The evaluation of our method through the use of the prototype system is another future work.

REFERENCES


5 RELATED WORKS

Ben Achour proposed guidance for correcting scenarios, based on a set of rules (Achour, 1998). These rules aim at the clarification, completion and conceptualization of scenarios, and help the scenario author to improve the scenarios until an acceptable level in terms of the scenario models. Ben Achour's rules can only check whether the scenarios are well written according to the scenario models. We propose generation methods of exceptional scenarios and alternative scenarios from a normal scenario.

Derek Cramp claimed the importance of alternative scenarios. He proposed a model to create alternative scenarios (Cramp et al., 1995). However, his model strongly depends on a specific domain.

Ian Alexander proposed a scenario-driven search method to find more exceptions (Alexander, 2000). In his approach, a model answer was prepared with knowledge of all exception cases identified by stakeholders. For each event, related exceptions are listed as a model answer. His model answer, however, strongly depends on a specific domain.

Neil Maiden et al. proposed classes of exceptions for use cases (Maiden et al, 1998). These classes are generic exceptions, permutations exceptions, permutation options, and problem exceptions. With these classes, alternative courses are generated. They proposed a generation method of alternative paths for each normal sequence from exception types for events and generic requirements with abnormal patterns (Sutcliff et al., 1998).

Our approach for generating scenarios with a differential scenario is independent of problem domains.
Keywords: System testing, use cases, generation of test cases.

Abstract: Use cases have become a widely used technique to define the functionality of a software system. This paper describes a new, formal and systematic approach for generating system test cases from use cases. This process has been designed specially for testing the system from the point of view of the actors, through its graphical user interfaces.

1 INTRODUCTION

System testing is a black-box technique which verifies the satisfaction of the requirements of the system under test (SUT) (Burnstein, 2003). Early testing is the generation of test cases in early development phases. This is not a new idea. Two surveys (Denger, 2003) and (Gutiérrez, 2004) (22 different approaches in total) expose that there are many lacks in the exiting approaches. One lack is the absence of a formal process and the absence of free available tools. Another lack is that approaches are not complete; this means that they describe how to generate partial test cases, mainly test actions, without describing other important elements such as test data, expected result, executable test scripts, or test coverage.

In a previous work it was described how to generate test cases from use cases for web application using existing approaches (Gutiérrez, 2005). This paper tries to resolve both lacks offering a formal approach for obtaining executable test scripts from use cases. It has been specially designed to be used in early development phases. It also uses UML and UML Testing Profile (OMG, 2002) (called UMLTP from now). Related works may be found in (Denger, 2003) and (Gutiérrez, 2004).

2 A PROCESS TO GENERATE TEST CASES FROM USE CASES

This test process is focused on testing use cases whose principal actor is human. A test case is composed of three elements: test action, test values and expected results. Test actions are the actions developed by the test case over the system under test (SUT). Test values are the information needed by the test case. Expected results are the responses of the system that allows evaluating whether the test is satisfied or failed. The results for this process are: test objectives, a set of test cases to verify each objective and test scripts. Test cases are expressed using models and graphical notation defined in UMLTP when possible.

2.1 Testing Models

The models used to store the information about test cases are: test objective model, test data model, interface model and event model.

1. Test objective models.

A test objective is an element named according to the description of what should be tested. The UMLTP does not define any notation to represent test objectives. Thus, we use activity diagrams.

A test objective is a path through the activity diagram. Test objectives might be automatically extracted from the activity diagram applying a coverage criterion, like all-edges and all-transitions. Every test objective will have at least one test case to verify it. An example is shown in table 4.

2. Test data model.

Test data model describes the structure and values of the test data. The first task is to identify operational variables (or simply variables) of a use case. An operational variable is an explicit input or output, an environmental condition or a representation of the
SUT (Binder, 1999). The domain of every variable is divided into data partitions. UMLTP uses class diagrams and stereotypes to describe the hierarchy of data partitions. After that, test values are generated for every data partition. Case study shows an example of data structures, partitions and test values in figure 4.

3. Interface model.
Our aim is to test the functionality throughout a graphical interface, not to test the graphical interface itself. The objective of this model is to describe the interface used for the test case to interact with the system. Since this process has been designed to be applied in early development phases, this model represents a high abstract description of the GUI. UML Testing Profile, and UML in general, does not include any specific notation for GUI, so it will be used class and object diagrams to represent the components and states of a GUI.

Figure 1: Example of components for interface models.

Figure 1 shows a class diagram with some elements from an interface model.

4. Event model.
Frequently, actor activities from a test objective are too abstract to be directly translated into a test script. It is proposed to build up an event model to address this complexity. A set of events describes how to perform actor activities identified in the test objective model. If an activity needs to supply information to the system, this information should have been defined in the test data model. This paper introduces a simple set of messages to express events. These messages are listed in table 1. Due to their simplicity, the messages might be easily extended. Event model also includes an assert message (table 1). This message is sent by the test case to itself to verify an attribute of the GUI. This message allows codifying the expected results into a test script, an example is shown in case study.

<table>
<thead>
<tr>
<th>Message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClickOn(component)</td>
<td>Perform a one-click event over the indicated GUI component.</td>
</tr>
<tr>
<td>Screen(screen)</td>
<td>Search for the indicate GUI screen and set the focus over it.</td>
</tr>
<tr>
<td>SetField(field, value)</td>
<td>Set the indicated value into the field object.</td>
</tr>
<tr>
<td>Assert(component. attribute, value)</td>
<td>Verify that the attribute of the component indicated matches with the value.</td>
</tr>
</tbody>
</table>

Table 1: Messages for event model.

2.2 Steps to Generate Test Cases

We suggest a process of six steps to generate test models and to obtain executable test scripts. These steps are shown in the activity diagram in figure 2 and described in the following paragraphs.

The first step is to build up a test objective model from a use case, as described in point 2.1. The second one is to build up the test data model as described in point 2.1. In the third step, test cases are generated combining test objectives with test values. The number of test cases is determined by the test objectives and the different partitions for the variables involved in that test objective. In the fourth step, interface model is generated, as described in point 2.1. In the fifth step event model is generated. Finally, event messages, test values and assertions are translated into test scripts, completed with test harness (Binder, 1999) and executed over the real SUT.
3 CASE STUDY

This section applies the process described in section 2 over a real system to generate test cases. The system under test is an implementation of a classic notepad. The use case selected to generate test cases is Open File (table 2).

<table>
<thead>
<tr>
<th>Description</th>
<th>Load document from file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precondition</td>
<td>No</td>
</tr>
</tbody>
</table>
| Main scenario | 1 User select “Open file” option.  
2 System asks for the file to open.  
3 User selects a file.  
4 System loads the file and shows the document. |
| Alternative / errors | 3 User may cancel the loading operation at any time.  
4 If file does not exist or there is an error, system shows an error message. |
| Post condition | No. |

A full coverage for the use case is selected. This means that at least one test case for every identified test objective will be generated.

3.1 Generation of Test Objectives

First of all, the test objective model is built (figure 3). Activities 01 and 03 are developed by the user and activities 02, 04, 04.1 and 04.2 are performed by the system. Step 4 (table 4) has been divided into activities 04, 04.1 and 04.2, and due to their results they may be different if there is an error when opening the file.

3.2 Generation of Test Values

Firstly the variables involved in the use case are identified. Test objective model in figure 3 reveals that there are, at least, two variables (the same number as decision nodes). Variables and domains are resumed in table 3.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>User-Option</td>
<td>Options available for the user: load file or cancel.</td>
</tr>
<tr>
<td>File</td>
<td>File to open</td>
</tr>
</tbody>
</table>

User-Option is a variable of an enumerated type. However, File is a variable of a complex type. For the testing purpose there must be known, at least, the name of the file, its content and its attributes.

3.3 Build Test Cases

A test case is a test objective with a concrete value for its variables. Variables and their partitions are added to the test objectives, as shown in table 4.

| Test objectives | 1 01, 02, 03(01: Without-Errors, op01:LoadOption), 04, 04.1  
2 01, 02, 03(02: With-Errors, op02:LoadOption), 04, 04.1  
3 01, 02, 03(03: *, op03:CancelOption), 04, 04.1 |

3.4 Generate Interface Model

It is assumed that the system under test is not built yet. So, it is generated an abstract description of the user interface with the minimum set of components to perform the use case.

Studying the use case, we realize that there are three screens involved: the main screen, where user actor clicks on open option, the file selection screen,
where user actor selects the file to open, and the error screen where system shows the error message, if any.

The interface model is shown in the object diagram in figure 5.

![Interface model](image)

Figure 5: Interface model.

### 3.5 Generate Event Models and Expected Results

First, each user activity is refined using messages listed in table 1 and the user interface, defined in point 3.4. The event model to verify the main scenario is described as UML sequence diagram in figure 6.

![Event model](image)

Figure 6: Event model.

Validation actions are implemented using the assert proposition shown in table 1 and activities diagrams as proposed in the UMLTP. Due their simplicity, the have been omitted.

The process has ended. There have been generated test actions (shown in the event model) and expected results (shown in the event model too) that commits our test objectives (shown in the test objective model). Up to now, we have not needed the design or the code of the system.

### 3.6 Building Test Scripts

The information obtained in the points before, might be automatically translated into executable test scripts. Details of the implementation and test tool are needed to perform this step. It is used a real implementation of the notepad, called Stylepad, to generate test scripts. The Stylepad is distributed in the Java Developer Kit. It has been used an open source tool called Abbot to codify executable scripts.

### 4 CONCLUSIONS

This paper has shown a process for the early-testing of use cases. Although this process has been designed to test use cases from the perspective of human actors, it can be also used to test other actors. This process can be applied in early development stages. In fact, in case study described in section 3, all test cases have been generated before choosing a real implementation to test.

### REFERENCES


Distributed and Parallel Systems
Full Papers
Algorithmic skeletons are predefined components for parallel programming. We will present a skeleton for branch & bound problems for MIMD machines with distributed memory. This skeleton is based on a distributed work pool. We discuss two variants, one with supply-driven work distribution and one with demand-driven work distribution. This approach is compared to a simple branch & bound skeleton with a centralized work pool, which has been used in a previous version of our skeleton library Muesli. Based on experimental results for two example applications, namely the $n$-puzzle and the traveling salesman problem, we show that the distributed work pool is clearly better and enables good runtimes and in particular scalability. Moreover, we discuss some implementation aspects such as termination detection as well as overlapping computation and communication.

1 INTRODUCTION

Today, parallel programming of MIMD machines with distributed memory is mostly based on message-passing libraries such as MPI (W. Gropp, 1999; MPI, 2006). The resulting low programming level is error-prone and time consuming. Thus, many approaches have been suggested, which provide a higher level of abstraction and an easier program development. One such approach is based on so-called algorithmic skeletons (Cole, 1989; Cole, 2006), i.e. typical patterns for parallel programming which are often offered to the user as higher-order functions. By providing application-specific parameters to these functions, the user can adapt an application independent skeleton to the considered parallel application. (S)he does not have to worry about low-level implementation details such as sending and receiving messages. Since the skeletons are efficiently implemented, the resulting parallel application can be almost as efficient as one based on low-level message passing.

Algorithmic skeletons can be roughly divided into data parallel and task parallel ones. Data-parallel skeletons (see e.g. (R. Bisseling, 2005; G. H. Botorog, 1996; G. H. Botorog, 1998; H. Kuchen, 1994; Kuchen, 2002; Kuchen, 2004)) process a distributed data structure such as a distributed array or matrix as a whole, e.g. by applying a function to every element or by rotating or permuting its elements. Task-parallel skeletons (A. Benoit, 2005; Cole, 2004; Hofstedt, 1998; H. Kuchen, 2002; Kuchen, 2002; Kuchen, 2004; Pelagatti, 2003) construct a system of processes communicating via streams of data. Such a system is mostly generated by nesting typical building blocks such as farms and pipelines. In the present paper, we will focus on a particular task-parallel skeleton, namely a branch & bound skeleton.

Branch & bound (G.L. Nemhauser, 1999) is a well-known and frequently applied approach to solve certain optimization problems, among them integer and mixed-integer linear optimization problems (G.L. Nemhauser, 1999) and the well-known traveling salesman problem (J.D.C. Little, 1963). Many practically important but NP-hard planning problems can be formulated as (mixed) integer optimization problems, e.g. production planning, crew scheduling, and vehicle routing. Branch & bound is often the only practically successful approach to solve these problems exactly. In the sequel we will assume without loss of generality that an optimization problem consists of finding a solution value which minimizes an objective function while observing a system of constraints. The main idea of branch & bound is the fol-
lowing. A problem is recursively divided into subproblems and lower bounds for the optimal solution of each subproblem are computed. If a solution of a (sub)problem is found, it is also a solution of the overall problem. Then, all other subproblems can be discarded, whose corresponding lower bounds are greater than the value of the solution. Subproblems with smaller lower bounds still have to considered recursively.

Only little related work on algorithmic skeletons for branch & bound can be found in the literature (E. Alba, 2002; F. Almeida, 2001; I. Dorta, 2003; Hofstedt, 1998). However, in the corresponding literature there is no discussion of different designs. The MaLLBa implementation is based on a master/worker scheme and it uses a central queue (rather than a heap) for storing problems. The master distributes problems to workers and receives their solutions and generated subproblems. On a shared memory machine this approach can work well. We will show in the sequel that a master/worker approach is less suited to handle branch & bound problems on distributed memory machines. In a previous version of the Muesli skeleton library, a branch & bound skeleton with a centralized work pool has bee used, too (H. Kuchen, 2002). Hofstedt outlines a B&B skeleton with a distributed work pool. Here, work is only shared, if a local work pool is empty. Thus, worthwhile problems are not propagated quickly and their investigation is concentrated on a few workers only.

The rest of this paper is structured as follows. In Section 2, we recall, how branch & bound algorithms can be used to solve optimization problems. In Section 3, we introduce different designs of branch & bound skeletons in the framework of the skeleton library Muesli (Kuchen, 2002; Kuchen, 2004; Kuchen, 2006). After describing the simple centralized design considered in (H. Kuchen, 2002), we will focus on a design with a distributed work pool. Section 4 contains experimental results demonstrating the strengths and weaknesses of the different designs. In Section 5, we conclude and point out future work.

2 BRANCH & BOUND

Branch & bound algorithms are general methods used for solving difficult combinatorial optimization problems. In this section, we illustrate the main principles of branch & bound algorithms using the 8-puzzle, a simplified version of the well-known 15-puzzle (Quinn, 1994), as example. A branch & bound algorithm searches the complete solution space of a given problem for the best solution. Due to the exponentially increasing number of feasible solutions, their explicit enumeration is often impossible in practice. However, the knowledge about the currently best solution, which is called incumbent, and the use of bounds for the function to be optimized enables the algorithm to search parts of the solution space only implicitly. During the solution process, a pool of yet unexplored subsets of the solution space, called the work pool, describes the current status of the search. Initially there is only one subset, namely the complete solution space, and the best solution found so far is infinity. The unexplored subsets are represented as nodes in a dynamically generated search tree, which initially only contains the root, and each iteration of the branch & bound algorithm processes one such node. This tree is called the state-space tree. Each node in the state-space tree has associated data, called its description, which can be used to determine, whether it represents a solution and whether it has any successors. A branch & bound problem is solved by applying a small set of basic rules. While the signature of these rules is always the same, the concrete formulation of the rules is problem dependent. Starting from a given initial problem, subproblems with pairwise disjoint state spaces are generated using an appropriate branching rule. A generated subproblem can be estimated applying a bounding rule. Using a selection rule, the subproblem to be branched from next is chosen from the work pool. Last but not least subproblems with non-optimal or inadmissible solutions can be eliminated during the computation using an elimination rule. The sequence of the application of these rules may vary according to the strategy chosen for selecting the next node to process (J. Clausen, 1999). As an example of the branch and bound technique, consider the 8-puzzle (Quinn, 1994). Figure 1 illustrates the goal state of the 8-puzzle and the first three levels of the state-space tree.

The 8-puzzle consists of eight tiles, numbered 1 through 8, arranged on a 3 × 3 board. Eight positions on the board contain exactly one tile and the remaining position is empty. The objective of the puzzle is to repeatedly fill the hole with a tile adjacent to it in horizontal or vertical direction, until the tiles are in row major order. The aim is to solve the puzzle in the least number of moves.

The branching rule describes, how to split a problem represented by a given initial board into subproblems represented by the boards resulting after all valid moves. A minimum number of tile moves needed to solve the puzzle can be estimated by adding the number of tile moves made so far to the Manhattan distance between the current position of each tile and its goal position. The computation of this lower bound is described by the bounding rule.

The state-space tree represents all possible boards that can be reached from the initial board. One way to solve this puzzle is to pursue a breadth first search or a depth first search of the state-space tree until the
sorted board is discovered. However, we can often reach the goal faster by selecting the node with the best lower bound to branch from. This selection rule corresponds to a best-first search strategy. Other selection rules such as a variant of depth-first search are discussed in (J. Clausen, 1999; Y. Shinano, 1995; Y. Shinano, 1997).

Branch & bound algorithms can be parallelized at a low or at a high level. In case of a low-level parallelization, the sequential algorithm is taken as a starting point and just the computation of the lower bound, the selection of the subproblem to branch from next, and/or the application of the elimination rule are performed by several processes in a data parallel way. The overall behavior of such a parallel algorithm resembles of the sequential algorithm.

In case of a high-level parallelization, the effects and consequences of the parallelism are not restricted to a particular part of the algorithm, but influence the algorithm as a whole. Several iterations of the main loop are performed in a task-parallel way, such that the state-space tree is explored in a different (non-deterministic!) order than in the sequential algorithm.

3 BRANCH & BOUND SKELETONS

In this section, we will consider different implementation and design issues of branch & bound skeletons. For the most interesting distributed design, several work distribution strategies are discussed and compared with respect to scalability, overhead, and performance. Moreover, a corresponding termination detection algorithm is presented.

A B&B skeleton is based on one or more branch & bound algorithms and offers them to the user as predefined parallel components. Parallel branch & bound algorithms can be classified depending on the organization of the work pool. A central, distributed, and hybrid organization can be distinguished. In the MaLLBa project, a central work pool is used (F. Almeida, 2001; I. Dorta, 2003). Hofstedt (Hofstedt, 1998) sketches a distributed scheme, where work is only delegated, if a local work pool is empty. Shinano et al. (Y. Shinano, 1995; Y. Shinano, 1997) and Xu et al. (Y. Xu, 2005) describe hybrid approaches. A more detailed classification can be found in (Trienekens, 1990), where also complete and partial knowledge bases, different strategies for the use of knowledge and the division of work as well as the chosen synchronicity of processes are distinguished.

Moreover, different selection rules can be fixed. Here, we use the classical best-first strategy. Let us mention that this can be used to simulate other strategies such as the depth-first approach suggested by Clausen and Perregaard (J. Clausen, 1999). The bounding function just has to depend on the depth in the state-space tree.

We will consider the skeletons in the context of the skeleton library Muesli (Kuchen, 2002; Kuchen, 2004; Kuchen, 2006). Muesli is based on MPI (W. Gropp, 1999; MPI, 2006) internally in order to inherit its platform independence.

3.1 Design with a Centralized Work Pool Manager

The simplest approach is a kind of the master/worker design as depicted in Figure 2. The work pool is maintained by the master, which distributes problems to the workers and receives solutions and subproblems from them. The approach taken in a previous version of the skeleton library Muesli is based on this centralized design. When a worker receives a problem, it either solves it or decomposes it into subproblems and computes a lower bound for each of the subproblems. The work pool is organized as a heap, and the subproblem with the best lower bound at the time is stored in its root. Idle workers are served with new problems taken from the root. This selection
The code fragment in Fig. 3 illustrates the application of our skeleton in the context of the Muesli library. It constructs the process topology shown in Fig. 2.

```c++
int main(int argc, char* argv[]) {
    InitSkeletons(argc,argv);
    // step 1: create a process topology
    Initial<Problem> initial(generateProblem);
    Filter<Problem,Problem> filter(generateCases,1);
    BranchAndBound<Problem> bnb(filter,n,
                                betterThan,isSolution);
    Final<Problem> final(fin);
    Pipe pipe(initial,bnb,final);
    // step 2: start process topology
    pipe.start();
    TerminateSkeletons();
}
```

Figure 3: Example application using a branch and bound skeleton with centralized work pool manager.

In a first step the process topology is created using C++ constructors. The process topology consists of an initial process, a branch & bound process, and a final process connected by a pipeline skeleton. The initial process is parameterized with a `generateProblem` method returning the initial optimization problem that is to be solved. The filter process represents a worker. The passed function `generateCases` describes, how to branch & bound subproblems. The constructor `BranchAndBound` produces `n` copies of the worker and connects them to the internal work pool manager (which is not visible to the user). `bool betterThan(Problem x1, Problem x2)` has to deliver `true`, if the lower (upper) bound for the best solution of problem `x1` is better than the lower (upper) bound for the best solution of problem `x2` in case of a minimization (maximization) problem. This function is used internally for the work pool organization. The function `bool isSolution(Problem x)` can be used to discover, whether its argument `x` is a solution or not. The final process receives and processes the optimal solution. Problems and solutions are encoded by the same type `Problem`.

The advantage of a single central work pool maintained by the master is that it provides a good overall picture of the work still to be done. This makes it easy to provide each worker with a good subproblem to branch from and to prune the work pool. Moreover, the termination of the workers is easy to implement, because the master knows about all idle workers at any time, and the best solution can be detected easily. The disadvantage is that accessing the work pool tends to be a bottleneck, as the work pool can only be accessed by one worker at a time. This may result in high idle times on the workers’ site. Another disadvantage is that the master/worker approach incurs high communication costs, since each subproblem is sent from its producer to the master and propagated to its processing worker. If the master decides to eliminate a received subproblem, time is wasted for its transmission. Moreover, the communication time required to send a problem to a worker and to receive in return some subproblems may be greater than the time needed to do the computation locally. The master’s limited memory capacity for maintaining the work pool is another disadvantage of this architecture.

As we will see in the next subsection, these disadvantages can be avoided by a distributed maintenance of the work pool. However, this design requires a suitable scheme for distributing subproblems and some distributed termination detection.

---

1 Remember that task-parallel skeletons can be nested.
3.2 Distributed Work Pool

Figure 5 illustrates the design of the distributed branch and bound (DBB) skeleton provided by the Muesli skeleton library. It consists of a set of peer solvers, which exchange problems, solutions, and (possibly) load information. Several topologies for connecting the solvers are possible. For small numbers of processors, a ring topology can be used, since it enables an easy termination detection. For larger numbers of processors, topologies like torus or hypercube may lead to a faster propagation of work from hot spots to idle processors. For simplicity, we will assume a ring topology in the sequel. Compared to more complicated topologies the ring also simplifies the dynamic adaption of the number of workers in case that more or less computation capacity has to be devoted to the branch & bound skeleton within the overall computation. This (not yet implemented) feature will enable a well-balanced overall computation.

In our example, \( n = 5 \) solvers are used. Each solver maintains its own local work pool and has one entrance and one exit. Exactly one of the solvers, called the master solver, serves as an entrance to the DBB-skeleton and receives new optimization problems from the predecessor. Any of the \( n \) solvers may deliver the detected optimal solution to the successor of the branch & bound skeleton in the overall process topology. All solvers know each other for a fast distribution of newly detected best solutions 2. If the skeleton only consists of a single solver neither communication nor distributed termination detection are necessary. In this case all communication parts as well as the distributed termination detection algorithm are bypassed to speed up the computation.

The code fragment in Fig. 4 shows an example application of our distributed B&B skeleton. It constructs the process topology depicted in Fig. 5. Work request messages are only sent when using a demand-driven work distribution.

The construction of the process topology resembles that in the previous example. Instead of a filter a BBSolver process is used as a worker. In addition to the betterThan and isSolution function two other argument functions are passed to the constructor, namely a branch and a bound function. The constructor DistributedBB produces \( n \) copies of the solver. One of the solvers is automatically chosen as the master solver.

As described in the previous section, a task-parallel skeleton consumes a stream of input values and produces a stream of output values. If the master solver receives a new optimization problem, the communication with the predecessor is blocked until the received problem is solved. This ensures that the skeleton processes only one optimization problem at a time. There are different variants for the initialization of parallel branch & bound algorithms with the objective of providing each worker with a certain amount of work within the start-up phase. Ideally, the work load is distributed equally to all workers. However, the work load is hard to predict without any domain knowledge. For this reason the skeleton uses the most common approach, namely root initialization, i.e. the root of the state space tree is inserted into the local work pool of the master solver. Subproblems are distributed according to the load balancing scheme applied by the solvers. This initialization has the advantage that it is very easy to implement and no additional code is necessary. Other initialization strategies are discussed in the literature. A good survey can be found in (Henrich, 1994a).

Each worker repeatedly executes two phases: a communication phase and a solution phase. Let us first consider the communication phase. In order to avoid that computation time is wasted with the solution of irrelevant subproblems, it is essential to spread and process new best solutions as quickly as possible. For this reason, we distinguish problem messages and incumbent messages. Each solver first checks for arriving incumbents with MPI_Testsome. If it has received new incumbents, the solver stores the best and discards the others. Moreover, it removes subproblems whose lower bound is worse than the incumbent from the work pool. Then, it checks for arriving subproblems and stores them in the work pool, if their lower bounds are better than the incumbent.

The solution phase starts with selecting an unexamined subproblem from the work pool. As in the master/worker design, the work pool is organized as a heap and the selection rule implements a best-first search strategy. The selected problem is decomposed into \( m \) subproblems by applying branch. For each

```c
int main(int argc, char* argv)
{
    InitSkeletons(argc,argv);
    // step 1: create a process topology
    Initial<Problem> initial(generateProblem);
    BBSolver<Problem> solver("ring",branch,bound,
        betterThan,isSolution);
    DistributedBB<Problem> bnb =
        DistributedBB<Problem>{solver,n};
    Final<Problem> final(fin);
    Pipe pipe(initial,bnb,final);
    // step 2: start process topology
    pipe.start();
    TerminateSkeletons();
}
```

Figure 4: Task parallel example application of a fully distributed Branch and Bound skeleton.

---

2Thus, the topology is in fact a kind of wheel with spokes rather than a ring.
of the subproblems, we proceed as follows. First, we check, whether it is solved. If a new best solution is detected, we update the local incumbent and broadcast it. A worse solution is discarded. Finally, if the subproblem is not yet solved, the bound function is applied and the subproblem is stored in the work pool (see Fig. 5).

### 3.3 Load Distribution and Knowledge Sharing

Since the work pools of the different solvers, grow and shrink differently, some load balancing mechanism is required. Many global and local load distribution schemes have been studied in the literature (Henrich, 1994b; Henrich, 1995; R. Lüling, 1992; N. Mahapatra, 1998; Sanders, 1998; A. Shina, 1992) and many of them are suited in the context of a distributed branch & bound skeleton. Here, we will focus on two local load balancing schemes, a supply- and a demand-driven one. The local schemes avoid the larger overhead of a global scheme. On the other hand, they need more time to distribute work over long distances.

With the simple supply-driven scheme, each worker sends in each $i$th iteration its second best problem to its right neighbor in the ring topology. It always processes the best problem itself, in order to avoid communication overhead compared to the sequential algorithm. The supply driven approach has the advantage that it distributes work slightly more quickly than a demand driven approach, since there is no need for work requests. This may be beneficial in the beginning of the computation. A major disadvantage of this approach is that many subproblems are transmitted in vain, since they will be sooner or later discarded at their destination due to better incumbents, in particular for small $i$. Thus, high communication costs are caused.

The demand-driven approach distributes load only in case that a neighbor requests it. In our case, a neighbor sends the lower bound of the best problem in its work pool (see Fig. 5). If this value is worse than the lower bound of the second best problem of the worker receiving this information, it is interpreted as a work request and a problem is transmitted to the neighbor. In case that the work pool of the neighbor is empty, the information message indicates this fact rather than transmitting a lower bound. An information message is sent every $i$th iteration of the main loop. In order to avoid flooding the network with "empty work pool" messages, such messages are never sent twice. If the receiver of an "empty work pool message" is idle, too, it stores this request and serves it as soon as possible. The advantage of this algorithm is that distributing load only occurs, if it is necessary and beneficial. The overhead of sending load information messages is very low due to their small sizes. For small $i$ the overhead is bigger, but idle processors get work more quickly.

### 3.4 Termination Detection

In the distributed setting, it is harder to detect that the computation has finished and the optimal solution has been found. The termination detection algorithm used in the DBB-skeleton is a variant of Dijkstra’s algorithm outlined in (Quinn, 1994). Our implementation utilizes the specific property of MPI that the order in which messages are received from a sender $S$ is always equal to the order in which messages were sent by $S$. This characteristic can be used for the purpose of termination detection in connection with local load distribution strategies as described above.

As mentioned, we arrange the workers in a ring topology, since this renders the termination detection particularly easy and simplifies the dynamic addition and removal of workers. For a small number of processors (as in our system), the large diameter of the
Let \( n \) be the number of solvers of the DBB-skeleton. When the master solver receives a new optimization problem, it initializes the termination detection by sending a token along the ring in the same direction as the load is distributed. The token only consists of an \( \text{int} \) value. Initially, the token has the value \( n \). If a solver receives a new subproblem, this event is noted by setting a flag to \( \text{true} \). On arrival of a token the solver uses the rules stated by the following pseudo code:

```c
IF (workpool is empty AND flag == false)  
    token := token - 1;
IF (workpool is not empty OR flag == true) {  
    token := n; flag := false; }
IF (token > 0) send token to successor;
IF (token == 0) computation is finished;
```

Only if all workers are idle, the token is decremented by every worker and the computation is finished. No more problems can be in the network, since the token cannot overtake other messages on its way. Note that this algorithm only works for load balancing strategies which send load in the same direction as the token.

### 4 EXPERIMENTAL RESULTS

We have tested the different versions of the branch & bound skeleton experimentally on an IBM workstation cluster (ZIV, 2006) using up to 16 Intel Xeon EM64T processors with 3.6 GHz, 1 MB L2 Cache, and 4 GB memory, connected by a Myrinet (Myricom, 2006). As example applications we have considered the \( n \)-puzzle as explained in section 2 as well as a parallel version of the traveling salesman problem (TSP) algorithm by Little et al. (J.D.C. Little, 1963). Both differ w.r.t. the quality of their bounding functions and hence in the number of considered irrelevant subproblems.

The presented B&B algorithm for the \( n \)-puzzle has a rather bad bounding function based on the Manhattan distance of each tile to its destination. It is bad, since the computed lower bounds are often much below the value of the best solution. As a consequence, the best-first search strategy is not very effective and the number of problems considered by the parallel skeleton differs enormously over several runs with the same inputs. This number largely depends on the fact whether a subproblem leading to the optimal solution is picked up early or late. Note that the parallel algorithm behaves non-deterministically in the way the search-space tree is explored. In order to get reliable results, we have repeated each run 100 times and computed the average runtimes.

The goal of the TSP is to find the shortest round trip through \( n \) cities. Little’s algorithm represents each problem by its residual adjacency matrix, a set of chosen edges representing a partially completed tour, and a lower bound on the length of any full tour, which can be generated by extending the given partial tour. New problems are produced by selecting a key edge and generating two new problems, in which the chosen edge is included and excluded from the emerging tour, respectively. The key edge is selected based on the impact that the exclusion of the edge will have on the lower bound. The lower bounds are computed based on the fact that each city has to be entered and left once and that consequently one value in every row and column of the adjacency matrix has to be picked. The processing of a problem mainly requires three passes through the adjacency matrix.

The TSP algorithm computes rather precise lower bounds. Thus, the best-first strategy works fine, and the parallel implementation based on Quinn’s formulation of the algorithm (Quinn, 1994) considers only very few problems more than the sequential algorithm, as explained below.
Consequently, the runtimes were relatively similar over several runs with the same parameters. For the TSP, we have used a real world 16 city map taken and adapted from (Reinelt, 1991) and 300 randomly generated 30 city maps. The real world map has much more sub-tours with similar lengths. Thus, proportionally more subproblems are processed which do not lead to the optimal solution than for the artificial map, where the best solution is found more easily.

When comparing the supply- and the demand-driven approach (see Figure 6 and the 3rd columns of Tables 1, 2), we notice that, as expected, the demand driven scheme is better, since it produces less communication overhead. The fact that the problems are distributed slightly slower causes no serious performance penalty.

For the supply driven scheme, we have used an optimal number $i$ for the amount of iterations that a worker waits before delegating a problem to a neighbor. If $i$ is chosen too large, important problems will not spread out fast enough. If $i$ is too small, the communication overhead will be too large. We found that the optimal value for $i$ depends on the application problem and on the number of workers. If the number of workers increases, $i$ has to be increased as well. In our experiments, the optimal values for $i$ were ranging between 2 and 20 for up to 8 workers.

As expected, we see that for the centralized B&B skeleton the work pool manager quickly becomes a bottleneck and it has difficulties to keep more than 2 workers busy (see Figures 6, 8 and Table 3). This is due to the fact that the amount of computations done for a problem is linear in the size of the problem, just as the communication complexity for sending and receiving a problem. Thus, relatively little is gained by delegating a problem to a worker. The work pool manager has to spend only little work less for transmitting the problem than its processing would require. This property is typical for virtually all practically relevant branch & bound problems we are aware of. It has the important consequence that a centralized work pool manager does not work well for branch & bound on distributed memory machines. Also note that the centralized scheme needs one more processor, the work pool manager, than the distributed one rendering this approach even less attractive.

Both variants of the design with a distributed work pool do not have these drawbacks (see Figures 6, 7, 8 and Tables 1, 2). Here, the communication overhead is much smaller. Each worker fetches most problems from its own work pool, such that they require no communication. This is particularly true for the demand driven approach. This scheme has the advantage that after some start-up phase, in which all workers are supplied with problems, there is relatively little communication and the workers mainly process locally available problems. This is essential for achieving good runtimes and speedups. We anticipate that this insight not only applies to branch & bound but also to other skeletons with a similar characteristic.

Table 1: Distribution of problems for the 16 city TSP using a distributed work pool and demand driven work distribution.

<table>
<thead>
<tr>
<th>#workers</th>
<th>runtime (s)</th>
<th># considered problems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

Figure 8: Speedups for 24-puzzle using the central work pool manager and the distributed work pool with demand-driven work distribution depending on the number of workers.

Table 2: Distribution of problems for the 16 city TSP using a distributed work pool and supply driven work distribution.

<table>
<thead>
<tr>
<th>#workers</th>
<th>runtime (s)</th>
<th># considered problems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

Table 3: Distribution of problems for the 16 city TSP using a central work pool manager.

<table>
<thead>
<tr>
<th>#workers</th>
<th>runtime (s)</th>
<th># considered problems</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
such as divide & conquer and other search skeletons. We are currently working on experimental results supporting this claim.

Interestingly we could even observe slightly super-linear speedups for the 30 city TSPs. They can be explained by the fact that a parallel B&B algorithm may tackle important subproblems earlier than the sequential one, since it processes the state-space tree in a different order (T. Lai, 1984).

It is clear that a parallel B&B algorithm will typically consider more problems than a corresponding sequential one, since it eagerly processes several problems in parallel, which would be discarded in the sequential case, since their lower bounds are worse than a detected solution. Interestingly for both considered example applications, TSP and $n$-puzzle, the corresponding overhead was very small and only few additional problems have been processed by the parallel implementation (see the 3rd columns of Tables 1, 2, 3). For instance, for the 16 city TSP no more than $274002 - 263019 = 10983$ additional problems are processed by the parallel algorithm; this is less than 4.2%. This is essential for achieving reasonable speedups.

As an implementation detail of the centralized approach let us mention that it is important that the work pool manager receives in each iteration all available subproblems and solutions from the workers rather than just one of them. The reason is that MPI_Waitany (used internally) is unfair and that an overloaded work pool manager will hence almost exclusively communicate with a small number of workers. If a starving worker has an important subproblem (one that leads to the optimal solution) or a good solution, which it is not able to deliver to the work pool manager, this will cause very bad runtimes.

Another implementation detail of the centralized approach concerns the amount of buffering. In order to be able to overlap computation and communication, it is a good idea that the work pool manager not only sends one problem to each worker and then waits for the results, but that it sends $n$ problems such that the worker can directly tackle the next problem after finishing the previous one. Here it turned out that one has to be careful not to choose $n$ too large, since then problems which would otherwise be discarded due to appearing better incumbents will be processed (in vain). In our experiments, $n = 2$ was a good choice.

5 CONCLUSION

We have considered two different implementation schemes for the branch & bound skeleton. Besides a simple approach with a central work pool manager, we have investigated a scheme with a distributed work pool. As our analysis and experimental results show, the communication overhead is high for the centralized approach and the work pool manager quickly becomes a bottleneck, in particular, if the number of computation steps for each problem grows linearly with the problem size, as it is the case for virtually all practically relevant branch & bound problems. Thus, the centralized scheme does not work well in practice.

On the other hand, our scheme with a distributed work pool works fine and provides good runtimes and scalability. The latter is not trivial, as discussed e.g. in the book of Quinn (Quinn, 1994), since parallel B&B algorithms tend to process an increasing number of irrelevant problems the more processors are employed. In particular, the demand-driven design works well due to its low communication overhead.

For the supply-driven approach, we have investigated how often a problem should be propagated to a neighbor. Depending on the application and the number of workers, we have observed the best runtimes, if a problem was delegated between every 2nd and every 20th iteration.

We are not aware of any previous comparison of different implementation schemes of branch & bound skeletons for MIMD machines with distributed memory in the literature. In the MaLLBa project (E. Alba, 2002; F. Almeida, 2001), a branch & bound skeleton based on a master/worker approach and a queue for storing subproblems has been developed. But as we pointed out above, this scheme is more suitable for shared memory machines than for distributed memory machines. Hofstedt (Hofstedt, 1998) sketches a B&B skeleton with a distributed work pool. Here, work is only delegated, if a local work pool is empty. A quick propagation of “interesting” subproblems are missing. According to our experience, this leads to a suboptimal behavior. Moreover, Hofstedt gives only few experimental results based on reduction steps in a functional programming setting rather than actual runtimes and speedups.

As future work, we intend to investigate alternative implementation schemes of skeletons for other search algorithms and for divide & conquer.

REFERENCES


299


PARALLEL PROCESSING OF "GROUP-BY JOIN" QUERIES ON
SHARED NOTHING MACHINES

M. Al Hajj Hassan and M. Bamha
LIFO, Université d’Orléans
B.P. 6759, 45067 Orléans Cedex 2, France
{mohamad.alhajjhassan,mostafa.bamha}@univ-orleans.fr

Keywords: PDBMS, Parallel joins, Data skew, Join product skew, GroupBy-Join queries, BSP cost model.

Abstract: SQL queries involving join and group-by operations are frequently used in many decision support applications. In these applications, the size of the input relations is usually very large, so the parallelization of these queries is highly recommended in order to obtain a desirable response time. The main drawbacks of the presented parallel algorithms that treat this kind of queries are that they are very sensitive to data skew and involve expansive communication and Input/Output costs in the evaluation of the join operation. In this paper, we present an algorithm that minimizes the communication cost by performing the group-by operation before redistribution where only tuples that will be present in the join result are redistributed. In addition, it evaluates the query without the need of materializing the result of the join operation and thus reducing the Input/Output cost of join intermediate results. The performance of this algorithm is analyzed using the scalable and portable BSP (Bulk Synchronous Parallel) cost model which predicts a near-linear speed-up even for highly skewed data.

1 INTRODUCTION

Data warehousing, On-Line Analytical Processing (OLAP) and other multidimensional analysis technologies have been employed by data analysts to extract interesting information from large database systems in order to improve the business performance and help the organisations in decision making. In these applications, aggregate queries are widely used to summarize large volume of data which may be the result of the join of several tables containing billions of records (Datta et al., 1998; Chaudhuri and Shim, 1994). The main difficulty in such applications is that the result of these analytical queries must be obtained interactively (Datta et al., 1998; Tsois and Sel-lis, 2003) despite the huge volume of data in warehouses and their rapid growth especially in OLAP systems (Datta et al., 1998). For this reason, parallel processing of these queries is highly recommended in order to obtain acceptable response time (Bamha, 2005). Research has shown that join, which is one of the most expansive operations in DBMS, is parallelizable with near-linear speed-up only in ideal cases (Bamha and Hains, 2000). However, data skew degrades the performance of parallel systems (Bamha and Hains, 1999; Bamha and Hains, 2000; Seetha and Yu, 1990; Hua and Lee, 1991; Wolf et al., 1994; De-Witt et al., 1992). Thus, effective parallel algorithms that evenly distribute the load among processors and minimizes the inter-site communication must be employed in parallel and distributed systems in order to obtain acceptable performance.

In traditional algorithms that treat "GroupBy-Join" queries, join operations are performed in the first step and then the group-by operation (Chaudhuri and Shim, 1994; Yan and Larson, 1994). But the response time of these queries is significantly reduced if the group-by operation is performed before the join (Chaudhuri and Shim, 1994), because group-by reduces the size of the relations thus minimizing the join and data redistribution costs. Several algorithms that perform the group-by operation before the join operation were presented in the literature (Shatdal and Naughton, 1995; Taniar et al., 2000; Taniar and Rahayu, 2001; Yan and Larson, 1994). In the "Early Distribution Schema" algorithm presented in (Taniar and Rahayu, 2001), all the tuples of the tables are redistributed before applying the join operations. GroupBy-Join queries are queries involving group-by and join operations.
or the group-by operations, thus the communication cost in this algorithm is very high. However, the cost of its join operation is reduced because the group-by is performed before the expansive join operation. In the second algorithm, “Early GroupBy Scheme” (Taniar and Rahayu, 2001), the group-by operation is performed before the distribution and the join operations thus reducing the volume of data. But in this algorithm, all the tuples of the group-by results are redistributed even if they do not contribute in the join result. This is a drawback, because in some cases only few tuples of relations formed of million of tuples contribute in the join operation, thus the distribution of all these tuples is useless.

These algorithms fully materialize the intermediate results of the join operations. This is a significant drawback because the size of the result of this operation is generally large with respect to the size of the input relations. In addition, the Input/Output cost in these algorithms is very high where it is reasonable to assume that the output relation cannot fit in the main memory of each processor, so it must be reread in order to evaluate the aggregate function.

In this paper, we present a new parallel algorithm used to evaluate “GroupBy-Join” queries on Shared Nothing machines (a multiprocessors machine where each processor has its own memory and disks (DeWitt and Gray, 1992)). In this algorithm, we do not materialize the join operation as in the traditional algorithms where the join operation is evaluated first and then the group-by and aggregate functions (Yan and Larson, 1994). So the Input/Output cost is minimal because we do not need to save the huge volume of data that results from the join operation.

We also use the histograms of both relations in order to find the tuples which will be present in the join result. After finding these tuples, we apply on them the grouping and aggregate function, in each processor, before performing the join. Using our approach, we reduce the size of data and communication costs to minimum. It is proved in (Bamha and Hains, 2000; Bamha and Hains, 1999), using the BSP model, that histogram management has a negligible cost when compared to the gain it provides in reducing the communication cost. In addition, our algorithm avoids the problem of data skew because the hashing functions are only applied on histograms and not on input relations.

The performance of this algorithm is analyzed using the scalable and portable BSP cost model (Skillicorn et al., 1997) which predicts for our algorithm a near-linear speed-up even for highly skewed data.

The rest of the paper is organized as follows. In section 3, we present the BSP cost model used to evaluate the processing time of the different phases of the algorithm. In section 3, we give an overview of different computation methods of “GroupBy-Join” queries. In section 4, we describe our algorithm. We then conclude in section 5.

2 THE BSP COST MODEL

Bulk-Synchronous Parallel (BSP) cost model is a programming model introduced by L. Valiant (Valiant, 1990) to offer a high degree of abstraction like PRAM models and yet allow portable and predictable performance on a wide variety of multi-processor architectures (Bisseling, 2004; Skillicorn et al., 1997). A BSP computer contains a set of processor-memory pairs, a communication network allowing inter-processor delivery of messages and a global synchronization unit which executes collective requests for a synchronization barrier. Its performance is characterized by 3 parameters expressed as multiples of the local processing speed:

- the number of processor-memory pairs \( p \),
- the time \( t \) required for a global synchronization,
- the time \( g \) for collectively delivering a 1-relation (communication phase where each processor receives/sends at most one word). The network is assumed to deliver an \( h\)-relation in time \( g + h\) for any arity \( b \).

\[
\begin{align*}
&\text{Figure 1: A BSP superstep.} \\
&P1 P2 P3 \ldots Pp
\end{align*}
\]

A BSP program is executed as a sequence of supersteps, each one divided into (at most) three successive and logically disjoint phases. In the first phase each processor uses only its local data to perform sequential computations and to request data transfers to/from other nodes. In the second phase the network delivers the requested data transfers and in the third phase a global synchronization barrier occurs, making the transferred data available for the next superstep. The execution time of a superstep \( s \) is thus the sum of the maximal local processing time of, the data delivery
time and of the global synchronization time:

\[\text{Time}(s) = \max_{i \text{ processor}} w_i(s) + \max_{i \text{ processor}} h_i(s) \cdot g + l\]

where \(w_i(s)\) is the local processing time on processor \(i\) during superstep \(s\) and \(h_i(s) = \max\{h_i(s), h_i(s)\}\), where \(h_i(s)\) (resp. \(h_i(s)\)) is the number of words transmitted (resp. received) by processor \(i\) during superstep \(s\). The execution time, \(\sum_s \text{Time}(s)\), of a BSP program composed of \(S\) supersteps is therefore a sum of 3 terms: \(W + H + g + S + l\) where \(W = \sum_i \max w_i(s)\) and \(H = \sum_i \max h_i(s)\). In general \(W, H\) and \(S\) are functions of \(p\) and of the size of data \(n\), or (as in the present application) of more complex parameters like data skew and histogram sizes. To minimize execution time of a BSP algorithm, design must jointly minimize the number \(S\) of supersteps and the total volume \(h\) (resp. \(W\)) and imbalance \(h^{(s)}\) (resp. \(W^{(s)}\)) of communication (resp. local computation).

### 3 COMPUTATION OF "GROUP-BY JOIN" QUERIES

In DBMS, the aggregate functions can be applied on the tuples of a single table, but in most SQL queries, they are applied on the output of the join of multiple relations. In the later case, we can distinguish two types of "GroupBy-Join" queries. We will illustrate these two types using the following example.

In this example, we have three relations that represent respectively Suppliers, Products and the quantity of a product shipped by a supplier in a specific date.

\[\text{SUPPLIER} \quad \text{PRODUCT} \quad \text{SHIPMENT}\]

**Query 1**

\[
\text{Select } p.Pid, \text{SUM} (\text{Quantity}) \\
\text{From PRODUCT as } p, \text{SHIPMENT as } s \\
\text{Where } p.Pid = s.Pid \\
\text{Group By } p.Pid
\]

**Query 2**

\[
\text{Select } p.Category, \text{SUM} (\text{Quantity}) \\
\text{From PRODUCT as } p, \text{SHIPMENT as } s \\
\text{Where } p.Pid = s.Pid \\
\text{Group By } p.Category
\]

The purpose of Query 1 is to find the total quantity of every product shipped by all the suppliers, while that of Query 2 is to find the total amount of every category of product shipped by all the suppliers. The difference between Query 1 and Query 2 lies in the group-by and join attributes. In Query 1, the join attribute (\(P\text{id}\)) and the group-by attribute are the same. In this case, it is preferable to carry out the group-by operation first and then the join operation (Taniar et al., 2000; Taniar and Rahayu, 2001), because the group-by operation reduces the size of the relations to be joined. As a consequence, applying the group-by operation before the join operation in PDBMS\(^2\) results in a huge gain in the communication cost and the execution time of the "GroupBy-Join" queries.

In the contrary, this can not be applied on Query 2, because the join attribute (\(P\text{id}\)) is different from the group-by attribute (\(\text{category}\)).

In this paper, we focus on "GroupBy-Join" queries when the join attributes are part of the group-by attributes. In our algorithm, we succeeded to redistribute only tuples that will be present in the join result after applying the aggregate function. Therefore, the communication cost is reduced to minimum.

### 4 PRESENTED ALGORITHM

In this section, we present a detailed description of our parallel algorithm used to evaluate "GroupBy-Join" queries when the join attributes are part of the group-by attributes. We assume that the relation \(R\) (resp. \(S\)) is partitioned among processors by horizontal fragmentation and the fragments \(R_i\) for \(i = 1, ..., p\) are almost of the same size on each processor, i.e. \(|R_i| \approx \frac{|R|}{p}\) where \(p\) is the number of processors.

For simplicity of description and without loss of generality, we consider that the query has only one join attribute \(x\) and that the group-by attribute set consists of \(x\), an attribute \(y\) of \(R\) and another attribute \(z\) of \(S\). We also assume that the aggregate function \(f\) is applied on the values of the attribute \(u\) of \(S\). So the treated query is the following:

\[
\text{Select } R.x, R.y, S.z, f(S.u) \\
\text{From } R, S \\
\text{Where } R.x = S.x \\
\text{Group By } R.x, R.y, S.z
\]

In the rest of this paper, we use the following notation for each relation \(T \in \{R, S\}\):

- \(T_i\) denotes the fragment of relation \(T\) placed on processor \(i\),
- \(Hist^{w_i}(T)\) denotes the histogram\(^3\) of relation \(T\) with respect to the attribute \(w\), i.e. a list of pairs

\(^2\)PDBMS : Parallel DataBase Management Systems.

\(^3\)Histograms are implemented as a balanced tree (B-tree): a data structure that maintains an ordered set of data to allow efficient search and insert operations.
of blocks $S_i$ are created on the fly while scanning relation $S_i$ in parallel, on each processor $i$, by applying the aggregate function $f$ on every group of tuples having identical values of the couple of attributes $(x, z)$. At the same time, the local histograms $Hist^x_y(R_i)_{i=1,...,p}$ are also created.

(In this algorithm the aggregate function may be $\text{MAX}, \text{MIN}, \text{SUM}$ or $\text{COUNT}$. For the aggregate $\text{AVG}$ a similar algorithm that merges the $\text{COUNT}$ and the $\text{SUM}$ algorithms is applied).

In principle, this phase costs:

$$Time_{\text{phase}1} = O(c_{i/a} \cdot \max_{i=1,...,p} (|R_i| + |S_i|)).$$

**Phase 2: Creating the histogram of $R \bowtie S$**

The first step in this phase is to create the histograms $Hist^x_y(R)$ and $Hist^x_y(S)$ by a parallel hashing of the histograms $Hist^x_y(R_i)$ and $Hist^x_y(S_i)$. After hashing, each processor $i$ merges the messages it received to constitute $Hist^x_y(R)$ and $Hist^x_y(S)$. While merging, processor $i$ also retains a trace of the network layout of the values $d$ of the attribute $x$ in its $Hist^x_y(R)$ (resp. $Hist^x_y(S)$): this is nothing but the collection of messages it has just received. This information will help in forming the communication templates in phase 3.

The cost of redistribution and merging step is (cf. to proposition 1 in (Bamha and Hains, 2005)):

$$Time_{\text{phase}2,a} = O \left( \min \left( g \cdot |Hist^x_y(R)| + |Hist^x_y(S)|, g \cdot \frac{|R|}{p} + \frac{|R|}{p} \right) + |Hist^x_y(S)|, g \cdot \frac{|S|}{p} + \frac{|S|}{p} + l \right).$$

where $g$ is the BSP communication parameter and $l$ the cost of a barrier of synchronisation.

We recall that, in the above equation, for a relation $T \in \{R, S\}$, the term $\min(g \cdot |Hist^x_y(T)| + |Hist^x_y(T)|, g \cdot \frac{|T|}{p} + \frac{|T|}{p})$ is the necessary time to compute $Hist^x_y(T)_{i=1,...,p}$ starting from the local histograms $Hist^x_y(T_i)_{i=1,...,p}$.

The histogram $Hist^x_y(R \bowtie S)$ is then computed on each processor $i$ by intersecting $Hist^x_y(R)$ and $Hist^x_y(S)$ in time:

$$Time_{\text{phase}2,b} = O \left( \max_{i=1,...,p} (\min(||Hist^x_y(R)|, ||Hist^x_y(S)||)) \right).$$

---

$AGGR^w_{f,u}(T) \equiv \text{AGGR}^w_{U,s}(T)$ is implemented as a balanced tree (B-tree).

---

(v, u), where $n_u \neq 0$ is the number of tuples of relation $T$ having the value $v$ for the attribute $u$. The histogram is often much smaller and never larger than the relation it describes,

- $Hist^w(T_i)$ denotes the histogram of fragment $T_i$,
- $Hist^w(T)$ is processor $i$’s fragment of the histogram of $T$,
- $Hist^w(T)(v)$ is the frequency ($n_v$) of value $v$ in relation $T$,
- $Hist^w(T_i)(v)$ is the frequency of value $v$ in sub-relation $T_i$,
- $AGGR^w_{f,u}(T)$ is the result of applying the aggregate function $f$ on the values of the aggregate attribute $u$ of every group of tuples of $T$ having identical values of the group-by attribute $w$. $AGGR^w_{f,u}(T)$ is formed of a list of tuples having the form $(v, f_v)$ where $f_v$ is the result of applying the aggregate function on the group of tuples having value $v$ for the attribute $w$ (when $w$ may be formed of more than one attribute),
- $AGGR^w_{f,u}(T_i)$ denotes the result of applying the aggregate function on the attribute $u$ of the fragment $T_i$,
- $AGGR^w_{f,u}(T)$ is processor $i$’s fragment of the result of applying the aggregate function on $T$,
- $AGGR^w_{f,u}(T)(v)$ is the result $f_u$ of the aggregate function of the group of tuples having value $v$ for the group-by attribute $w$ in relation $T$,
- $AGGR^w_{f,u}(T_i)(v)$ is the result $f_u$ of the aggregate function of the group of tuples having value $v$ for the group-by attribute $w$ in sub-relation $T_i$,
- $|T|$ denotes the number of tuples of relation $T$; and
- $|T|$ denotes the size (expressed in bytes or number of pages) of relation $T$.

The algorithm proceeds in four phases. We will give an upper bound of the execution time of each superstep using BSP cost model. The notation $O(\ldots)$ hides only small constant factors: they depend only on the program implementation but neither on data nor on the BSP machine parameters.

**Phase 1: Creating local histograms**

In this phase, the local histograms $Hist^x_y(R_i)_{i=1,...,p}$ (resp. $Hist^x_y(S_i)_{i=1,...,p}$) of blocks $R_i$ (resp. $S_i$) are created in parallel by a scan of the fragment $R_i$ (resp. $S_i$), on processor $i$, in time $c_{i/a} \cdot \max_{i=1,...,p} |R_i|$ (resp. $c_{i/a} \cdot \max_{i=1,...,p} |S_i|$) where $c_{i/a}$ is the cost of writing/reading a page of data from disk.

In addition, the local fragments $AGGR^y_{x,i}(S_i)_{i=1,...,p}$

---

$\text{ICSOFT 2006 - INTERNATIONAL CONFERENCE ON SOFTWARE AND DATA TECHNOLOGIES}$

304
The total cost of this phase is:
\[
T_{\text{phase2}} = T_{\text{phase2.a}} + T_{\text{phase2.b}}
\]
\[
O\left(\min\left(g \ast |\text{Hist}^x(R)| + |\text{Hist}^x(R)|, g \ast \frac{|R|}{p} + \frac{|R|}{p}\right) + \min\left(g \ast |\text{Hist}^x(S)|, g \ast \frac{|S|}{p} + \frac{|S|}{p}\right) + \max_{i=1,\ldots,p}\left(\min\{|\text{Hist}^x_i(R)|, |\text{Hist}^x(S)|\} + 1\right)\right).
\]

**Phase 3: Data redistribution**

In order to reduce the communication cost, only tuples of $\text{Hist}^{x,y}(R)$ and $\text{AGGR}^{x,y}_{f,u}(S)$ that will be present in the join result will be redistributed. To this end, we first compute on each processor $j$ the intersections $\overline{\text{Hist}}^{(j)x}(R_i) = \text{Hist}^{(j)x}(R_i) \cap \text{Hist}(R \bowtie S)$ and $\overline{\text{Hist}}^{(j)y}(S_i) = \text{Hist}^{(j)y}(S_i) \cap \text{Hist}(R \bowtie S)$ for $i = 1, \ldots, p$ where $\text{Hist}^{(j)x}(R_i)$ (resp. $\text{Hist}^{(j)y}(S_i)$) is the fragment of $\text{Hist}^x(R_i)$ (resp. $\text{Hist}^y(S_i)$) which was sent by processor $i$ to processor $j$ in the second phase.

The cost of this step is:
\[
O(\sum_i |\text{Hist}^{(j)x}(R_i)| + \sum_i |\text{Hist}^{(j)y}(S_i)|).
\]

We recall that,
\[
\sum_i |\text{Hist}^{(j)x}(R_i)| = \lvert \cup_i \text{Hist}^{(j)x}(R_i)\rvert \leq \min(|\text{Hist}^x(R)|, \frac{|R|}{p})
\]
and
\[
\sum_i |\text{Hist}^{(j)y}(S_i)| = \lvert \cup_i \text{Hist}^{(j)y}(S_i)\rvert \leq \min(|\text{Hist}^y(S)|, \frac{|S|}{p}),
\]
thus the total cost of this step is:
\[
T_{\text{phase3.a}} = O\left(\min(|\text{Hist}^x(R)|, \frac{|R|}{p}) + \min(|\text{Hist}^y(S)|, \frac{|S|}{p})\right).
\]

Now each processor $j$ sends each fragment $\overline{\text{Hist}}^{(j)x}(R_i)$ (resp. $\overline{\text{Hist}}^{(j)y}(S_i)$) to processor $i$. Thus, each processor $i$ receives
\[
\sum_j |\overline{\text{Hist}}^{(j)x}(R_i)| + \sum_j |\overline{\text{Hist}}^{(j)y}(S_i)|
\]
pages of data from the other processors.

In fact, $\text{Hist}^x(R_i) = \cup_j \overline{\text{Hist}}^{(j)x}(R_i)$ and $|\text{Hist}^x(R_i)| = \sum_j |\overline{\text{Hist}}^{(j)x}(R_i)| \geq \sum_i |\text{Hist}^{(i)x}(R_i) \cap \text{Hist}^x(R \bowtie S)|$, thus $|\text{Hist}^x(S_i)| \geq \sum_j |\overline{\text{Hist}}^{(j)y}(R_i)|$ (this also applies to $\text{Hist}^y(S_i)$).

Therefore, the total cost of this stage of communication is at most:
\[
T_{\text{phase3.b}} = O\left(g \ast (|\overline{\text{Hist}}^x(R)| + |\overline{\text{Hist}}^y(S)|) + 1\right).
\]

**Remark 1** $\cup_j \overline{\text{Hist}}^{(j)x}(R_i)$ is simply the intersection of $\text{Hist}^x(R_i)$ and the histogram $\text{Hist}^x(R \bowtie S)$ which will be noted:

\[
\overline{\text{Hist}}^x(R_i) = \cup_j \text{Hist}^{(j)x}(R_i) = \text{Hist}^x(R_i) \cap \text{Hist}^x(R \bowtie S).
\]

Hence $\overline{\text{Hist}}^x(R_i)$ is only the restriction of the fragment of $\text{Hist}^x(R_i)$ to values which will be present in the join of the relations $R$ and $S$. (this also applies to $\overline{\text{Hist}}^y(S_i)$).

Now, each processor obeys all the distributing orders it has received, so only tuples of $\text{Hist}^{x,y}(R_i) = \text{Hist}^{x,y}(R_i) \cap \overline{\text{Hist}}^x(R_i)$ and $\text{AGGR}^{x,y}_{f,u}(S_i) = \text{AGGR}^{x,y}_{f,u}(S_i) \cap \overline{\text{Hist}}^y(S_i)$ are redistributed.

To this end, we first evaluate $\text{Hist}^{x,y}(R_i)$ and $\text{AGGR}^{x,y}_{f,u}(S_i)$. The cost of this step is of order:
\[
T_{\text{phase3.c}} = O\left(\max_{i=1,\ldots,p}\left(|\text{Hist}^{x,y}(R_i)| + |\text{AGGR}^{x,y}_{f,u}(S_i)|\right)\right),
\]
which is the necessary time to traverse all the tuples of $\text{Hist}^{x,y}(R_i)$ and $\text{AGGR}^{x,y}_{f,u}(S_i)$ and access $\overline{\text{Hist}}^x(R_i)$ and $\overline{\text{Hist}}^y(S_i)$ respectively on each processor $i$.

Now, each processor $i$ distributes the tuples of $\text{Hist}^{x,y}(R_i)$ and $\text{AGGR}^{x,y}_{f,u}(S_i)$. After distribution, all the tuples of $\text{Hist}^{x,y}(R_i)$ and $\text{AGGR}^{x,y}_{f,u}(S_i)$ having the same values of the join attribute $x$ are stored on the same processor. So, each processor $i$ merges the blocks of data received from all the other processors in order to create $\overline{\text{Hist}}^x(R_i)$ and $\text{AGGR}^{x,y}_{f,u}(S_i)$.

The cost of distributing and merging the tuples is of order (cf. to proposition 1 in (Bamha and Hains, 2005)):
\[
T_{\text{phase3.d}} = O\left(\min\left(g \ast |\text{Hist}^{x,y}(R)| + |\text{Hist}^{x,y}(R)|, g \ast \frac{|R|}{p} + \frac{|R|}{p}\right) + \min\left(g \ast |\text{AGGR}^{x,y}_{f,u}(S)|, g \ast \frac{|S|}{p} + \frac{|S|}{p}\right) + 1\right),
\]
where the terms:
\[
\min\left(g \ast |\text{Hist}^{x,y}(R)| + |\text{Hist}^{x,y}(R)|, g \ast \frac{|R|}{p} + \frac{|R|}{p}\right)
\]
and
\[
\min\left(g \ast |\text{Hist}^{x,y}(R)| + |\text{Hist}^{x,y}(R)|, g \ast \frac{|S|}{p} + \frac{|S|}{p}\right)
\]
represent the necessary time to compute $\overline{\text{Hist}}^{x,y}(R)$ and $\text{AGGR}^{x,y}_{f,u}(S)$ starting from $\overline{\text{Hist}}^{x,y}(R)$ and $\overline{\text{Hist}}^{x,y}(S)$ respectively.
In this phase, we compute the global aggregate function.

Phase 4: Global computation of the aggregate function

In this phase, we compute the global aggregate function on each processor. We use the following algorithm where $AGGR_{f,u,i}(R \times S)$ holds the final result on each processor $i$. The tuples of $AGGR_{f,u,i}(R \times S)$ have the form $(x, y, z, v)$ where $v$ is the result of the aggregate function.

Par (on each node in parallel) $i = 1, ..., p$

$AGGR_{f,u,i}(R \times S) = NULL$  

For every tuple $t$ of relation $Hist^{x,y}(R)$ do

freq = $Hist^{x,y}(R)(t.x, t.y)$

For every entry $v_1 \in AGGR_{f,u,i}(S)(t.x, z)$ do

Insert a new tuple $(t.x, t.y, z, f(v_1, freq))$

into $AGGR_{f,u,i}(R \times S)$;

EndFor

EndFor

The time of this phase is:

$$Time_{phase4} = O \left( \min(g * ||Hist^{x,y}(R)|| + ||Hist^{x,y}(R)||, g * \frac{|R|}{p} + \frac{|R|}{p} + \frac{g}{p} \right)$$

Remark 2 In practice, the imbalance of the data related to the use of the hash functions can be due to:

- an intrinsic data imbalance which appears when some values of the join attribute appear more frequently than others. By definition a hash function maps tuples having the same join attribute values to the same processor. These is no way for a clever hash function to avoid load imbalance that result from these repeated values (DeWitt et al., 1992).

- But this case cannot arise here owing to the fact that histograms contains only distinct values of the join attribute and the hashing functions we use are always applied to histograms.

The global cost of evaluating the ”GroupBy-Join” queries is of order:

$$Time_{total} = O \left( \max_{i=1, ..., p} (|R_i| + |S_i|) \right)$$

Remark 3 In the traditional algorithms, the aggregate function is applied on the output of the join operation. The sequential evaluation of the ”GroupBy-Join” queries requires at least the following lower bound: $bound_{inf} = \Omega (c_{i/o} \ast (|R| + |S| + |R \times S|)).$

Parallel processing with $p$ processors requires therefore:

$$bound_{inf} = \frac{1}{p} \ast bound_{inf}.$$  

Using our approach in the evaluation of the ”GroupBy-Join” queries, we only redistribute tuples that will be effectively present in the ”GroupBy-Join” result, which reduces the communication cost to minimum. This algorithm has an asymptotic optimal complexity because all the terms in $Time_{total}$ are bounded by those of $bound_{inf}.$

5 Conclusion

The algorithm presented in this paper is used to evaluate the ”GroupBy-Join” queries on Shared Nothing machines when the join attributes are part of the group-by attributes. Our main contribution in this algorithm is that we do not need to materialize the
costly join operation which is necessary in all the other algorithms presented in the literature, thus we reduce its Input/Output cost. It also helps us to avoid the effect of data skew which may result from computing the intermediate join results and from redistributing all the tuples if AVS (Attribute Value Skew) exists in the relation. In addition, we partially evaluate the aggregate function before redistributing the data between processors or evaluating the join operation, because group-by and aggregate functions reduce the volume of data. To reduce the communication cost to minimum, we use the histograms to distribute only the tuples of the grouping result that will effectively be present in the output of the join operation. This algorithm is proved to have a near-linear speed-up, using the BSP cost model, even for highly skewed data. Our experience with the join operation (Bamha and Hains, 2000; Bamha and Hains, 1999; Bamha and Hains, 2005) is evidence that the above theoretical analysis is accurate in practice.

REFERENCES


IMPACT OF WRAPPED SYSTEM CALL MECHANISM ON COMMODORE PROCESSORS

Satoshi Yamada and Shigeru Kusakabe
Grad. School of Information Sci. & Electrical Eng., Kyushu University
6-10-1 Hakozaki, Higashi-ku, Fukuoka, 812-8581 Japan
Email: satoshi@ale.csce.kyushu-u.ac.jp, kusakabe@csce.kyushu-u.ac.jp

Hideo Taniguchi
Faculty of Engineering, Okayama University
3-1-1 Tsushima-naka, Okayama, 700-8530 Japan
Email: tani@cs.okayama-u.ac.jp

Keywords: System call, mode change, locality of reference.

Abstract: Split-phase style transactions separate issuing a request and receiving the result of an operation in different threads. We apply this style to system call mechanism so that a system call is split into several threads in order to cut off the mode changes from system call execution inside the kernel. This style of system call mechanism improves throughput, and is also useful in enhancing locality of reference. In this paper, we call this mechanism as Wrapped System Call (WSC) mechanism, and we evaluate the effectiveness of WSC on commodity processors. WSC mechanism can be effective even on commodity platforms which do not have explicit multithread support. We evaluate WSC mechanism based on a performance evaluation model by using a simplified benchmark. We also apply WSC mechanism to variants of cp program to observe the effect on the enhancement of locality of reference. When we apply WSC mechanism to cp program, the combination of our split-phase style system calls and our scheduling mechanism is effective in improving throughput by reducing mode changes and exploiting locality of reference.

1 INTRODUCTION

Although recent commodity processors are built based on a procedural sequential computation model, we believe some dataflow-like multithreading models are effective not only in supporting non-sequential programming models but also in achieving high throughput even on commodity processors. Based on this assumption, we are developing a programming environment, which is based on a dataflow-like fine-grain multithreading model(Culler et al., 1993). Our work also includes a dataflow-like multithread programming language and an operating system, CEFOS(Communication and Execution Fusion OS)(Kusakabe and et al, 1999).

In our dataflow-like multithreading model, we use a split-phase style system call mechanism in which a request of a system call and the receipt of the system call result are separated in different threads. Split-phase style transactions are useful in hiding latencies of unpredictably long operations in several situations. We apply this style to system calls and call as Wrapped System Call (WSC) mechanism. WSC mechanism is useful both in reducing overhead caused by system call mechanisms on commodity processors and in enhancing locality of reference.

In this paper, we evaluate the effectiveness of WSC mechanism on commodity processors. Section 2 introduces our operating system, CEFOS, and some of its features including WSC mechanism. Section 3 discusses the performance estimation and experimental benchmark results of WSC mechanism from the view point of system call overhead. Section 4 evaluates WSC mechanism for variants of cp program from the view point of locality of reference. We conclude WSC mechanism can reduce system call overhead and enhance locality of reference even on commodity platforms, which have no explicit support to dataflow-like multithreading.

2 SCHEDULING MECHANISMS IN CEFOS

2.1 CEFOS for Fine-Grained Multithreading

While running user programs under the control of an operating system like Unix, frequent context switches
### Table 1: Results of LMbench (Clock Cycles).

<table>
<thead>
<tr>
<th>Processor</th>
<th>null call</th>
<th>2p/0K</th>
<th>2p/16K</th>
<th>L1$</th>
<th>L2$</th>
<th>MainMem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celeron 500MHz</td>
<td>315</td>
<td>675</td>
<td>3235</td>
<td>3</td>
<td>11</td>
<td>93</td>
</tr>
<tr>
<td>Pentium4 2.53 GHz</td>
<td>1090</td>
<td>3298</td>
<td>5798</td>
<td>2</td>
<td>18</td>
<td>261</td>
</tr>
<tr>
<td>Intel Core Duo 1.6GHz</td>
<td>464</td>
<td>1327</td>
<td>2820</td>
<td>3</td>
<td>14</td>
<td>152</td>
</tr>
<tr>
<td>PowerPC G4 1GHz</td>
<td>200</td>
<td>788</td>
<td>2167</td>
<td>4</td>
<td>10</td>
<td>127</td>
</tr>
</tbody>
</table>

and communications between user processes and the kernel are performed behind the scenes. A system call requests a service of the kernel, and then voluntarily causes mode change. Activities involving operating system level operations are rather expensive on commodity platforms.

Table 1 shows the result of a micro-benchmark LMbench (McVoy and Staelin, 1996) on platforms with commodity processors and Linux. The row “null call” shows the overhead of a system call and the row “2p/0K” shows that of a process switch when we have two processes of zero KB context. Thus, the row “$x$ $y$” shows the overhead of a process switch for the pair of $x$ and $y$ which represent the number and the size of processes, respectively. The rows “L1$”, “L2$” and “MainMem” show the access latency for L1 cache, L2 cache and main memory, respectively.

As seen from Table 1, activities involving operating system level operations such as system calls and context switches are rather expensive on commodity platforms.

Therefore, one of the key issues to improve system throughput is to reduce the frequency of context switches and communications between user processes and the kernel. In order to address this issue, we employ mechanisms for efficient cooperation between the operating system kernel and user processes based on a dataflow-like multithreading model in CEFOS.

Figure 1 shows the outline of the architecture of CEFOS consisting of two layers: the external kernel in user mode and the internal kernel in supervisor mode. Internal kernel corresponds to the kernel of conventional operating systems. A process in CEFOS has a thread scheduler to schedule its ready threads.

A program in CEFOS consists of one or more partially ordered threads which may be fine-grained compared to conventional threads such as Pthreads. A thread in our system does not have a sleep state and we separate threads in a split-phase style at the points where we anticipate long latencies. Each thread is non-preemptive and runs to its completion without going through sleep states like Pthreads. While operations within a thread are executed based on a sequential model, threads can be flexibly scheduled as long as dependencies among threads are not violated.

A process in CEFOS has a thread scheduler and schedules its ready threads basically in the user-space. Since threads in CEFOS are a kind of user-level thread, we can control threads with small overhead. The external-kernel mechanism in CEFOS intermediates interaction between the kernel and thread schedulers in user processes. Although there exist some works on user level thread scheduling such as Capriccio (Behren and et al, 2003), our research differs in that we use fine-grain thread scheduling.

In order to simplify control structures, process control is only allowed at the points of thread switching. Threads in a process are not totally-ordered but partially-ordered, and we can introduce various scheduling mechanisms as long as the partial order relations among threads are not violated. Thus, CEFOS has scheduling mechanisms such as WSC mechanisms and Semi-Preemption mechanism.

### 2.2 Display Requests and Data (DRD) Mechanism

Operating systems use system calls or upcalls (E.A. Thomas and et al, 1991) for interactions between user programs and operating system kernel. System calls issue the demands of user processes through SVC and Trap instructions, and upcalls invoke specific functions of processes. The problem in these methods is overhead of context switches (Purohit and et al, 2003). We employ Display Requests and Data (DRD) mechanisms (Taniguchi, 2002) for cooperation between user processes and the kernel in CEFOS as we show below:

1. Each process and the kernel share a common memory area (CA).
2. Each process and the kernel display requests and necessary information on CA.
3. At some appropriate occasions, each process and the kernel check the requests and information displayed on CA, and change the control of its execution if necessary.

This DRD mechanism assists cooperation between processes and the kernel with small overhead. A sender or receiver of the request does not directly trigger the execution of request at the instance the request is generated. If the sender triggers directly the execution of receiver’s side, the system may suffer from
large overhead to switch. On the other hand, the system handles the request at its convenience with small overhead if we use DRD mechanism. For an extreme example, all requests from a process to the kernel are buffered and the kernel is called only when the process exhausts its ready threads.

The external kernel mechanism in CEFOS intermediates interaction between the internal kernel and thread schedulers in user processes by using this DRD mechanism. Thus, CEFOS realizes scheduling mechanisms such as WSC mechanism and Semi-Preemption mechanism by using DRD mechanism.

2.3 WSC Mechanism

WSC mechanism buffers system call requests from user programs until the number of the requests satisfies some threshold and then transfers the control to the internal kernel with a bucket of the buffered system call requests. Each system call request consists of four kinds of elements listed below.

- type of the system call
- arguments of the system call
- the address where the system call stores its result
- ID of the thread which the system call syncs after the execution

The buffered system calls are executed like a single large system call and each result of the original system calls is returned to the appropriate thread in the user process. Figure 2 illustrates the control flow in WSC mechanism, and each number in Figure 2 corresponds to the explanation below.

1. A thread requests a system call to External Kernel.
2. External Kernel buffers the request of system call to CA.
3. External Kernel checks whether the number of requests has reached the threshold. If the number of requests is less than the threshold, the thread scheduler is invoked to select the next thread from the ready threads in the process. If the number of request has reached the threshold, WSC mechanism sends the requests of system calls to the internal kernel to actually perform the system calls.
4. Internal Kernel accepts the requests of system calls and executes them one by one.
5. Internal Kernel stores the result of the system call to the address which Internal Kernel accepts as the third arguments of the system call. Also, Internal Kernel tells the thread, whose ID is accepted as the fourth argument, that it stores the result.
6. When Internal Kernel terminates executing all requests of system calls, External Kernel executes other threads. In other cases, WSC mechanism goes back to 3 and repeats this transaction.

WSC mechanism reduces overhead of system calls by decreasing the number of mode changes from user process to the kernel. Parameters and returned results of the buffered system calls under WSC mechanism are passed through CA of DRD to avoid frequent switches between the execution of user programs and that of the kernel.

3 EVALUATION: SYSTEM CALL OVERHEAD

We evaluate the effectiveness of WSC mechanism on commodity processors. The test platform is built by extending Linux 2.6.14 on commodity PCs.
First, we estimate the effectiveness of WSC mechanism by focusing on system call overhead. We compare the execution time of a program with normal system calls under the normal mechanism and that with split-phase system calls under WSC mechanism.

The total execution time of a program with N normal system calls under the normal mechanism, $T_{nor}$, is estimated as:

$$T_{nor} = T_{onor} + N \times (T_{sys} + T_{body}) + P_{nor} \quad (1)$$

where $T_{onor}$ is the execution time of the program portion excluding system calls under the execution of the normal system call mechanism, $T_{sys}$ is the setup and return cost of a single system call, and $T_{body}$ is the execution time of the actual body of the system call. In this estimation, we assume that we use the same system call and that there exist no penalties concerning memory hierarchies such as cache miss penalties and TLB miss penalties in $T_{onor}$ and $T_{body}$. $P_{nor}$ is the total penalties including cache miss penalties and TLB miss penalties during the execution of the normal system call mechanism.

Programs to which we can apply WSC mechanism are multithreaded and use split-phase style system calls. Additional thread management should be performed in this multithreaded program and we describe the overhead of this additional part as $T_{ek}$. $T_{ek}$ is estimated as:

$$T_{ek} = X \times T_{gen} + Y \times T_{sche} + Z \times T_{sync} \quad (2)$$

where X is the number of threads, $T_{gen}$ is the overhead to generate a single thread, Y is the number of times threads are scheduled, $T_{sche}$ is the overhead to schedule a thread, Z is the number of times synchronizations are tried and $T_{sync}$ is the overhead of a synchronization.

Although the execution of system call bodies will be aggregated, buffering system call request must be performed for each system call. We represent the overhead of buffering a single system call request as $T_{req}$. Thus, $T_{wsc}$, the total execution time of a program with N split-phase system calls under WSC mechanism, is estimated as:

$$T_{wsc} = T_{owsc} + T_{ek} + N \times T_{req} + \left[ \frac{N}{M} \right] \times T_{sys} + N \times T_{body} + P_{wsc} \quad (3)$$

where $T_{owsc}$ is the execution time of the program portion excluding system calls, M is the number of system calls to be buffered for a single WSC (i.e. WSC threshold) and $P_{wsc}$ is the total penalties concerning memory hierarchies including cache miss penalties and TLB miss penalties during the execution under WSC. We assume none of such penalties exists in $T_{owsc}$ as in the estimation for $T_{onor}$ and $T_{body}$.

$\Delta T$, the difference between the execution time under the normal mechanism and that of under CEFOS with WSC is estimated as:

$$\Delta T = T_{wsc} - T_{nor}$$

$$= (T_{owsc} - T_{onor}) + \{ T_{ek} + N \times T_{req} - (N - \left[ \frac{N}{M} \right]) \times T_{sys} \} + (P_{wsc} - P_{nor}) \quad (4)$$
We can say the performance is improved by WSC mechanism when $\Delta T < 0$. We estimate the value of $M$, the number of system calls to be buffered, to satisfy this condition. We assume the following conditions for the sake of simplicity: each program portion excluding system calls is the same, and each system call body is the same both in the normal version and in CEFOS version. Under these assumptions, we will only observe the difference of system call cost between the normal version and the CEFOS version. This assumption makes $T_{wsc} - T_{nor}$ and $P_{wsc} - P_{nor}$ amount to zero. We also assume $X$, $Y$ and $Z$ are equal to $N$. Thus, we can estimate the condition for $M$ to satisfy $\Delta T < 0$ as:

$$M > \frac{T_{sys}}{T_{sys} - (T_{gen} + T_{sche} + T_{sync} + T_{req})} \quad (5)$$

We measured each value in (5) in order to calculate the value of $M$ that satisfies the above condition as shown in Table 2 (we used the values of null call in Table 1 for $T_{sys}$).

The performance on Pentium4 2.53GHz and Intel Core Duo 1.66GHz will be improved when $M$ is larger than 1. The performance on Celeron 500MHz will be improved when $M$ is larger than 4. (Please note $M$ is a natural number)

### 3.2 Performance Evaluation Using getpid()

The above estimation assumed each system call body is the same both in the normal version and in CEFOS version for the sake of simplicity. In this subsection, we examine our estimation by using getpid() as a system call to meet such an assumption. We measured the number of clocks for a number of getpid() system calls using the hardware counter. We executed 128 getpid() system calls in our experiments. We changed the threshold of WSC as 1, 2, 4, 8, 16 and 32 for the WSC version. We also measured the total time of successive getpid() system calls under the normal system call convention in unchanged Linux.

Figure 3 shows the comparison results of clock cycles for getpid() system calls. The x-axis indicates the threshold of WSC and y-axis the ratio of clock cycles of WSC versions compared with clock cycles under the normal system call convention in unchanged Linux. The lower y value indicates the better result of WSC.

As seen from Figure 3, we have extra overhead when WSC threshold is 1, because of newly added load of $T_{gen}$, $T_{sche}$, $T_{sync}$ and $T_{req}$. However, we observe the effect of WSC when the threshold becomes 2 for Pentium4 2.53GHz and Intel Core Duo 1.66GHz and 4 for Celeron 500 MHz as we estimated in the previous estimation. The clock cycles in WSC versions are decreased as the threshold gets larger regardless of the processor type.

### 4 EVALUATION: LOCALITY OF REFERENCE

In the previous section, we evaluate the effectiveness of WSC mechanism in reducing overhead caused by system calls. In this section, we examine the effectiveness in exploiting locality of reference. We can expect high throughput when we can aggregate system calls which refer to the same code or data. The test platform is also built by extending Linux 2.6.14 on Pentium4 2.53 GHz.

#### 4.1 cp Program

We use modified cp programs to evaluate the effectiveness of WSC mechanism in exploit-
ing locality of reference. Figure 4 shows an overview of the control flow in cp program, and the symbols A, B and C in Figure 4 correspond to the ones in the explanation below.

A. one \texttt{open()} system call opens a file to read and the other \texttt{open()} system call opens another file to write, preparing a file descriptor for each file respectively.

B. \texttt{read()} system call reads to designated bytes from the file descriptor into buffer, and then \texttt{write()} system call writes to designated bytes to the file referenced by the file descriptor from buffer.

C. \texttt{close()} system call closes these files.

Thus, a cp program uses six system calls per transaction. We use a \texttt{cp} program called NORMAL version, which executes these six system calls in the order we show above, like \texttt{open()}, \texttt{open()}, \texttt{read()}, \texttt{write()}, \texttt{close()} and \texttt{close()}. We have to \texttt{open()} a file before executing \texttt{read()} or \texttt{write()}, and we have to specify the file descriptor, which is the result of \texttt{open()} system call, to execute \texttt{read()}, \texttt{write()} and \texttt{close()}. Therefore, we cannot simply wrap these six system calls. We have to wrap two \texttt{open()} system calls and other four system calls respectively. Because of the additional overhead of using WSC mechanism that we mentioned in Figure 3, we cannot expect the effect when applying WSC mechanism that we mentioned in Figure 3, we cannot expect the effect when applying WSC mechanism that we mentioned in Figure 3.

Thus, we consider doing multiple \texttt{cp}s in a program.

We use other four versions of \texttt{cp} program, and measure 11 portions of these 5 programs to observe: I. whether WSC mechanism is effective or not in \texttt{cp} programs in total,

II. the difference between the effect of wrapping single type of system calls and that of wrapping various types of system calls, and

III. the effect of wrapping system calls which have the same code but refer to different data.

Figure 5 shows these 5 programs and 11 portions.

"N" in Figure 5 is the number of \texttt{cp} transactions. Now, we explain each program and portion below. Then we explain why we choose these portions to examine the points of our interests above.

In Program 2 in Figure 5, we wrap every one of six kinds of system calls. We call this WSC+COLLECT version.

As a counterpart of this WSC+COLLECT, we also collect system calls of the same type in a block but execute the block with normal system call convention. We call this program as NORMAL+COLLECT version (Program 3 in Figure 5).

In addition, we implement WSC+RW and nor-

MAL+RW version (Program 4 and 5 in Figure 5), which change the order of \texttt{read()} and \texttt{write()} in WSC+COLLECT and NORMAL+COLLECT version.

Then, we show the explanation of 11 portions we measure. 1. from \texttt{open()} to \texttt{close()} of NORMAL.

2. from \texttt{open()} to \texttt{close()} of NORMAL+COLLECT.

3. from \texttt{open()} to \texttt{close()} of WSC+COLLECT.

4. from \texttt{open()} to \texttt{close()} of NORMAL+RW.

5. from \texttt{open()} to \texttt{close()} of WSC+RW.

6. \texttt{read()} and \texttt{write()} part of NORMAL+COLLECT.

7. \texttt{read()} and \texttt{write()} part of WSC+COLLECT.

8. \texttt{read()} and \texttt{write()} part of NORMAL+RW.

9. \texttt{read()} and \texttt{write()} part of WSC+RW.

10. only \texttt{write()} of NORMAL+COLLECT.

11. only \texttt{write()} of WSC+COLLECT.

We measured only \texttt{write()} in portion 10 and 11 to observe the effect of wrapping system calls which refer to different data. While \texttt{read()} system call contains disk access time, \texttt{write()} system call buffers access to the disk and enables us to observe the effect of WSC mechanism excluding disk access time. Also, we implemented NORMAL+RW and WSC+RW and measured portion 6, 7, 8 and 9 to observe the effect of wrapping two types of system calls together. Then, we measured the whole \texttt{cp} in portion 1, 2 and 3 to examine if WSC mechanism is effective or not in total. Also, we measured portion 4 and 5 to examine the influence of wrapping \texttt{read()} and \texttt{write()} system calls on \texttt{cp} total.

We measure clock cycles and the number of events such as L1 cache misses in every portion. From these results, we investigate how WSC mechanism effects locality of reference from the view point of I, II and III above.

4.2 Performance Evaluation

Table 3 shows the result of \texttt{cp} programs. In this case, WSC threshold is 8 and we do \texttt{cp} transactions 100 times, which means N in Figure 5 is 100. The numbers in the row “portion” correspond to the numbers of the explanation we show in subsection 4.1. The row “#clocks” shows clock cycles, the row “L2$” shows L2 cache miss counts and rows “ITLB” and “DTLB” show the walk counts for ITLB and DTB, respectively. We measured these events with a performance monitoring tool perfctr (Petterson, n.d.).
represents the execution of normal system calls
represents the execution of system calls to which we applied Wrapped System Call mechanism

Figure 5: Program and Portion we measure in cp program.

Figure 6: Comparison of clock cycles (write()).

In write() sections (portion 10 and 11), the clock cycles for WSC+COLLECT write() are less than NORMAL+COLLECT write() in about 0.12 million cycles, which is reduction to 83% in clock cycles. The reduction of this 0.12 million cycles by WSC mechanism is larger than the reduction estimated by using formula in section 3, which is about 0.072 million cycles for 100 write() system calls. We consider this improvement is achieved by enhanced locality of reference, therefore we measure the number of events concerning memory hierarchies. As we expected, we can see the reduction of L2 cache misses, ITLB walks and DTLB walks in WSC+COLLECT write() compared to NORMAL+COLLECT write(). Thus, we can say WSC mechanism is effective even when each system call refer to different data. We changed the threshold and measured portion 10 and 11 to compare the results with those of getpid(). Figure 6 shows the result, and we can see the same tendency as we see in Figure 3 that wrapping more than 2 system calls is effective in clock cycles in Pentium 4.

In read/write section (portion 6, 7, 8 and 9), we can see the effect of wrapping different system calls by comparing portion 6 with 7 and 8 with 9. Both clock cycles and number of events decrease in WSC version in both cases.

Finally, from portion 1, 2 and 3, we can say applying WSC mechanism to cp program is effective in total. When we compare portion 2 with 3 to ignore the difference of disk access pattern, the reduction in clock cycles is about 0.32 million cycles. As we see in write() system call, we can see the reduction of L2 cache misses, ITLB walks and DTLB walks. Therefore, we conclude that WSC mechanism for split-phase style system calls is effective in exploiting locality of reference. We can see the similar result from portion 4 with 5 and reach to the same conclusion.
Table 3: Results of cp program.

<table>
<thead>
<tr>
<th>portion</th>
<th>#clocks</th>
<th>L2$</th>
<th>ITLB</th>
<th>DTLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. NOR</td>
<td>2,884,325</td>
<td>7378</td>
<td>511</td>
<td>136</td>
</tr>
<tr>
<td>2. NOR+COLLECT</td>
<td>2,588,200</td>
<td>8800</td>
<td>187</td>
<td>207</td>
</tr>
<tr>
<td>3. WSC+COLLECT</td>
<td>2,262,523</td>
<td>7740</td>
<td>81</td>
<td>120</td>
</tr>
<tr>
<td>4. NOR+RW</td>
<td>2,625,804</td>
<td>8264</td>
<td>227</td>
<td>200</td>
</tr>
<tr>
<td>5. WSC+RW</td>
<td>2,431,758</td>
<td>8118</td>
<td>128</td>
<td>197</td>
</tr>
<tr>
<td>6. NOR+COLLECT read/write</td>
<td>1,090,608</td>
<td>4385</td>
<td>112</td>
<td>69</td>
</tr>
<tr>
<td>7. WSC+COLLECT read/write</td>
<td>876,703</td>
<td>3503</td>
<td>37</td>
<td>42</td>
</tr>
<tr>
<td>8. NOR+RW read/write</td>
<td>1,096,045</td>
<td>3875</td>
<td>130</td>
<td>72</td>
</tr>
<tr>
<td>9. WSC+RW read/write</td>
<td>985,227</td>
<td>3647</td>
<td>70</td>
<td>62</td>
</tr>
<tr>
<td>10. NOR+COLLECT write</td>
<td>686,883</td>
<td>1779</td>
<td>93</td>
<td>28</td>
</tr>
<tr>
<td>11. WSC+COLLECT write</td>
<td>569,206</td>
<td>1363</td>
<td>41</td>
<td>13</td>
</tr>
</tbody>
</table>

5 CONCLUSION

In this paper, we discussed our WSC mechanism in CEFOS. While CEFOS is based on a dataflow-like fine-grain multithreading model, WSC mechanism is effective in improving throughput even on commodity platforms which have no explicit support to dataflow-like fine-grain multithreading.

Today, many investigations have been made about utilizing multithreading processor, such as SMT. Many of them tackle with memory hierarchy problem because cache conflict often occurs under the condition where several threads run concurrently. One effective solution to this problem is improving the scheduling of thread, which is conventional Pthread, to utilize CPU resources more effectively (Snively and Tullsen, 2000). On the other hand, our work split conventional thread and control the thread in user process. Thus, we have more chances to schedule fine-grained threads more flexibly with smaller overhead.

In cp program, the combination of our split-phase style system calls and WSC mechanism is effective in improving throughput by reducing mode changes and penalties concerning memory hierarchies such as L2 cache misses and TLB walks.

Recently, the overhead of system call and context switch is increasing on commodity processors. Besides, we think the tendency continues that latency of memory access becomes bottleneck, which is coming from the gap between processor speed and memory speed. Therefore, we think WSC will be more effective in the future, which can reduce the overhead of system call and context switch and enhance the locality of reference. We believe this will contribute to higher throughput of internet server and large-scale computation in the future. Our future work includes collecting more data from other processors and exploiting the effect of SYSENTER/SYSEXIT command in x86 architecture.

REFERENCES


Short Papers
A HYBRID TOPOLOGY ARCHITECTURE FOR P2P FILE SHARING SYSTEMS

Department of Information Technologies and Communications, Polytechnic University of Cartagena, Campus Muralla del Mar, 30202, Cartagena, Spain
Email: {juanp.gea, josem.malgosa, pilar.manzanares, juanc.sanchez, antonio.guirado}@upct.es

Keywords: Peer-to-peer, structured networks, unstructured networks, application layer multicast.

Abstract: Over the Internet today, there has been much interest in emerging Peer-to-Peer (P2P) networks because they provide a good substrate for creating data sharing, content distribution, and application layer multicast applications. There are two classes of P2P overlay networks: structured and unstructured. Structured networks can efficiently locate items, but the searching process is not user friendly. Conversely, unstructured networks have efficient mechanisms to search for a content, but the lookup process does not take advantage of the distributed system nature. In this paper, we propose a hybrid structured and unstructured topology in order to take advantages of both kind of networks. In addition, our proposal guarantees that if a content is at any place in the network, it will be reachable with probability one. Simulation results show that the behaviour of the network is stable and that the network distributes the contents efficiently to avoid network congestion.

1 INTRODUCTION

The main characteristic of an overlay network is that all the computer terminals that shape it are organized defining a new network structure overlayed to the existent one. They are purely distributed systems, and can be used in a lot of interesting fields: for example, to transmit multicast traffic in a unicast network (like Internet), technique known as Application Layer Multicast (ALM). However, the most popular overlay networks are peer-to-peer (P2P) networks, commonly used to efficiently download large amounts of information. In this last scenario there are two types of P2P overlay networks: structured and unstructured.

The technical meaning of structured is that the P2P overlay network topology is tightly controlled. Such structured P2P systems have a property that consistently assigns uniform random NodeIDs to the set of peers into a large space of identifiers. With this identifier, the overlay network places the terminal in a specific position into a graph. On the other hand, in unstructured P2P networks the terminals are located in the overlay network by one (or several) rendez-vous terminals with network management functions.

Although unstructured P2P networks require the presence of one controller (rendez-vous) at least, they have the advantage that the information searching process supports complex queries (it is a similar methodology to that used to search for information in Google and supports keyword and phrases searching). That does not happen when the P2P network is structured. In this case the advantages are that it enables efficient discovery of data items and it doesn’t require any central controller. In addition, it is also much easier to reorganize when changes occur (registering and leaving terminals) and, consequently, the overlay network is more scalable and robust. Section 2 describes in depth the searching and location process of structured and unstructured networks.

In this work we try to design a file-sharing system that shares the advantages of both types of P2P networks. The users locate the contents in an unstructured way. If this search fails, the system will use an application layer multicast service (given by a structured P2P network), to locate the terminal that owns the searched information. It is necessary to remark that, with the system proposed in this work, the location of any existing content always success.

There are several proposals that try to support sophisticated search requirements, like (Garcés-Erice et al., 2003)(Mislove and Druschel, 2004)(Castro et al., 2002a). These proposals organize P2P overlays into a hierarchy, and they have a high degree of complexity.
The remainder of the paper is organized as follows: Section 2 describes the main characteristics of both, unstructured and structured networks. Section 3 describes the system proposed in this paper in detail. Section 4 summarizes the more relevant contributions of the proposed solution. Section 5 shows the simulation results and finally, Section 6 concludes the paper.

2 P2P OVERLAY NETWORKS

The topology in a P2P structured overlay network is algorithmly fixed. Both, the nodes and the contents, have assigned an identifier (NodeID and Key respectively) belonging to the same scope. These P2P systems use a hash function applied to a MAC or IP terminal address and to the data content respectively, to generate these identifiers. The overlay network organizes its peers into a graph that maps each data Key to a peer, so that content is placed not at random peers but at specified locations. This structured graph enables efficient discovery of data items using the given Keys: a lookup algorithm is defined and it is responsible for locating the content, knowing its identifier only. However, in its simple form, this class of systems does not support complex queries. They only support exact-match lookups: one needs to know the exact Key of a data item to locate the node(s) responsible for storing that item. In practice, however, P2P users often have only partial information for identifying these items and tend to submit broad queries (e.g., all the articles written by "John Smith") (Garcés-Erice et al., 2004). Some examples of P2P structured networks are: CAN, Chord, Tapestry, Kademlia and Viceroy (Stoica et al., 2003), (Zhao et al., 2004), (Maymounkov and Mazières, 2002).

Unstructured P2P networks are composed of nodes that are connected to the network without any previous knowledge of the topology. The terminals need to know beforehand the location of a central controller, also denoted rendez-vous point, responsible for including them within the overlay network and forward their contents list. The overlay networks organize peers in a random graph in a flat or hierarchical manner (e.g., Super-Peers layer). The search requests are sent to the rendez-vous node, and this evaluates the query locally on its own content, and supports complex queries. If the content is not located in the rendez-vous, most of the available networks use flooding or random walks or expanding-ring Time-To-Live (TTL) search on the graph to query content stored by overlay peers. This is inefficient because queries for content that are not widely replicated must be sent to a large fraction of peers, and there is no coupling between topology and data item’s location (Lua et al., 2005). Some examples of P2P unstructured networks are: Gnutella, FastTrack/Kazaa, BitTorrent and eDonkey 2000 (Lua et al., 2005).

In sum, for a human being, the searching process is easier in an unstructured network, since this is made using patterns of very high level (like in Google, for example). Nevertheless, there exists much inefficiency in the location process of the content. In structured networks, exactly the opposite happens: the location is quasi-immediate, but the searching process is more tedious.

3 DESCRIPTION OF THE SYSTEM

Our proposal tries to define a hybrid system. Therefore, the user can search contents using more or less general parameters and later choosing among the elements that satisfy the searching criterion that content which he wishes to download, like in unstructured networks. Nevertheless, the network will be organized in a structured way, which will facilitate the location of the contents. All the nodes are immersed in a structured overlay network (anyone of the previously mentioned types). In addition, the nodes divide automatically into different sub-groups, in a more or less uniform way, surrounding a rendez-vous node. This node has the best performances in terms of CPU, bandwidth and reliability (see Section 3.2). When searching for a content, the user will send the search parameters to its rendez-vous, and this will return information about who has the contents in this sub-group.

All the rendez-vous nodes of the network are going to be members of a multicast group defined within the same structured network. This way, if the search fails, the rendez-vous node will send the request to the rest of rendez-vous nodes in a multicast way. Fig. 1 describes the general architecture of the system.
3.1 Obtaining the Identifiers and Joining the General Network

Every node needs to obtain a NodeID. In this work, this identifier is obtained applying a hash function (MD5 or SHA-1) to its MAC or IP address. In the same way each node also needs to obtain a SubgroupID that identifies the sub-group to which the node is going to belong. We propose to use a previously well-known server to obtain this identifier. Each sub-group will have a maximum number of nodes, and the nodes will be assigned by order to each one of the sub-groups until completing their maximum capacity. When the existing sub-groups are completed new sub-groups will be created.

As is usual in any structured network a node needs to know at least one address of another node in the overlay network. The previous server can also provide this information. Finally, the node will have to link to the P2P overlay network, using the mechanism imposed by the structured network.

3.2 Joining the Sub-group

Each node of the sub-group will be able to establish a TCP connection with its rendez-vous node, and they will send their content list to it. Each sub-group is identified by a SubgroupID. Initially, a node looks for its rendez-vous. To do this, it uses the structured network to locate the node which NodeID fits with its SubgroupID. This node knows the IP address of the rendez-vous node of its sub-group. The last step consists of transmitting this information to the requester node. Note that in this way the system builds an unstructured network by using an underground structured network. In addition, this last property allows us to define the rendez-vous nodes dynamically and to guarantee the stability of the network throughout time.

3.3 Management of the Hierarchy

When the new node finds its rendez-vous, it notifies its resources of bandwidth and CPU. The rendez-vous nodes control the nodes that are linked to their subgroup and they form an ordered list of future rendez-vous candidates: the longer a node remains connected (and the better resources it has), the better candidate it becomes. This list is transmitted to all the members of the sub-group, and when the rendez-vous fails, the first node in the list becomes its successor. Later, it must inform all sub-group members that this node is now the new rendez-vous. Also, it must to modify this information in the node which NodeID fits with its SubgroupID.

3.4 Management of the Rendez-Vous Nodes

All the rendez-vous nodes are members of a multicast group defined at application level. When a node becomes rendez-vous, it must be linked to this multicast group, in order to spread the unsuccessful searches to the rest of sub-groups. Structured P2P networks can be used to implement an application layer multicast service, for example CAN-Multicast (Ratsanamy et al., 2001), Chord-Multicast (El-Ansary et al., 2003) and Scribe (Castro et al., 2002b). Each one uses a different P2P overlay and it can implement the multicast service using flooding (CAN-Multicast, Chord-Multicast) or the construction of a tree (Scribe). Any one of the previous methods provides an efficient mechanism to identify and to send messages to all the members of a group.

Our proposal uses Chord-Multicast. It is not necessary that the multicast process reaches all the group members before sending the searching results to the requester node. When one node responds affirmatively to a request it sends to the requester’s rendez-vous the coincidences of the search in its database. Next, this rendez-vous gives back immediately the IP address and the corresponding metadata to the requester node. Therefore, the requester node obtains the searching results as soon as possible.

3.5 Registering the Shared Files

In a similar way to KaZaA, when a node establishes connection with its rendez-vous it sends the metadata of those files that it wants to share. This allows the rendez-vous to maintain a data base including the identifiers of the files that all the nodes of the subgroup are sharing and the corresponding IP address of the node that contains them. The information sent by the node includes the name of the file, its size and its description.

3.6 Search

When a user wishes to make the search of certain content, his node sends a request on the TCP connection established with its rendez-vous. For each coincidence of the search in the data base, the rendez-vous gives back the IP address and the corresponding metadata.

If the search fails, the user has the possibility of asking for to its rendez-vous node that tries to contact with other rendez-vous. The identification of those nodes is simple, since all belong to the same application layer multicast group.
4 ADVANTAGES OF THE
SYSTEM

Next we are going to describe some of the contributions of the system proposed in this work. First, it is necessary to emphasize that all the nodes are assigned to a sub-group and not to a server. The nodes are able to automatically find the rendez-vous responsible for their sub-group.

It is also necessary to emphasize that this system is able to manage the heterogeneity of the network too. The most stable nodes and those with better benefits will become rendez-vous nodes, which will increase the network performances.

On the other hand, the application layer multicast service provides an effective way to share information among rendez-vous nodes. In this way the maintenance of multicast group is practically made in an automatic mode. In addition, this guarantees that any content in the network can be located by any user.

Finally, the searches will be made in a simple way, similar to those made in current unstructured file-sharing applications.

5 SIMULATIONS

One of the advantages of this system, commented previously, is that any content present in the network could be located by any user. Nevertheless, the searches of contents present in the same sub-group will be faster and more efficient than when the searches need to use other rendez-vous nodes.

There are several interesting parameters that is necessary to quantify. First, the probability that the requested content is registered in the rendez-vous of the node’s sub-group. Second, the evolution of the previous parameter throughout time. Since the users are making successive searches of contents in other sub-groups of the network, these automatically will be registered in their own rendez-vous node, increasing the value of this probability. Finally, it is also interesting to find out the average number of rendez-vous nodes that will be consulted in order to locate a content.

In order to quantify the previous parameters a simulator in C language has been programmed. The contents are classified in three classes based on the degree of interest that they can motivate in the users ("very interesting", "interesting" and "of little interest"). At the beginning, the available contents are distributed in a random way among all the nodes of the network. As has been mentioned before, the rendez-vous share information using a Chord-Multicast procedure.

The simulation results show the probability that a content is located in the same sub-group as the requester node, as well as the average and maximum number of rendez-vous consulted until content location. All these results are obtained based on the number of simulation iterations. In each one, all the nodes of the network ask for a content that they do not have.

Figures 2, 3 and 4 present the simulation results corresponding to a network with 12,800 different contents and 6,400 nodes, with 128 rendez-vous nodes.

Fig. 2 shows the probability that the content is in the same requester’s sub-group, for both the most interesting contents and for any content. It is observed that this probability grows as the number of iterations increases, but converging to a value of one, which assures that our system is stable. This also indicates that our architecture assures that, in a few steps, the contents will be equally distributed among all the sub-groups. It is also possible to observe that in the transitory, the probability of finding an interesting content in the rendez-vous increases more quickly than the probability of finding any content.

Fig. 3 shows the average number of rendez-vous consulted to find a content. It is observed that the number of consulted rendez-vous quickly decreases, and when the number of iterations reaches 500 this value converges to one, which indicates that the content is in the same sub-group as the requester node. This shows us that the load coming from other sub-groups is minimal. It is also observed that this parameter decreases more quickly in the case of the most interesting contents than in the case of other contents.

Finally, Fig. 4 shows, in linear scale, the maximum number of rendez-vous consulted to locate any content. This parameter oscillates a lot in the initial transitory, but when it finishes it converges to values near the unit, agreeing practically with the average number.

Next, we are going to check the effect that both the
number of contents and the number of rendez-vous nodes have on the probability that a content is located in the same requester’s sub-group. Figure 5 shows the previous probability but with 6,400 and 19,200 contents. It can be observed that when the number of contents in the network diminishes the probability of finding it in the same requester’s sub-group increases more quickly. On the other hand, when the number of contents in the network increases, a greater number of iterations is needed for the previous probability to reach the value of one.

Besides, Figure 6 shows the previous probability in a similar network but with 64 and 256 rendez-vous nodes. It can be observed that the effect of the number of rendez-vous on this probability is quite similar to the effect of the number of contents. When the number of rendez-vous diminishes, the probability of finding a content in the same requester’s sub-group increases more quickly. On the other hand, when the number of rendez-vous increases, a greater number of iterations is needed to obtain a probability close to one.

Next, we are going to compare the presented approach with the existing ones. In (Garcés-Erice et al., 2003), peers are organized into groups, and each group has its autonomous intra-group structured overlay network and lookup service. Groups are organized in a top-level structured overlay network. To find a peer that is responsible for a key, the top-level overlay first determines the group responsible for the key; the responsible group then uses its intra-group overlay to determine the specific peer that is responsible for the key. However, due to the use of structured networks this system does not support complex queries.
The main advantage of this system is to reduce the expected number of hops that are required for a lookup, but in any case this is bigger than in our system, since in steady state only one hop is required.

In (Mislove and Druschel, 2004), they call an instance of a structured overlay as an organizational ring. A multi-ring protocol stitches together the organizational rings and implements a global ring. Each ring has a globally unique ringID, which is known by all the members of the ring. Every search message carries, in addition to a target key, the ringID in which the key is stored. Then, the node forwards the message in the global ring to the group that corresponds to the desired ringID. When a key is inserted into a organizational ring, it is necessary that a special indirection record is inserted into the global ring that associates the key with the ringID of the organizational ring where key is stored. However, the expected number of hops that are required for a lookup is similar to the previous work.

Finally, in (Castro et al., 2002a), it is proposed the use of a universal ring, but it provides only bootstrap functionality while each service runs in a separate P2P overlay. The universal ring provides: an indexing service that enables users to find services of interest, a multicast service used to distributed software updates, a persistent store and distribution network that allows users to obtain the code needed to participate in a service’s overlay and a service to provide users with a contact node to join a service overlay.

6 CONCLUSIONS

This paper presents a hybrid P2P overlay network that makes easier for the user both the searching process and the content location. The simulation results show that in this type of networks the contents are distributed in a way that minimizes the overload on the rendez-vous nodes.

We have also verified that an increase of both the number of rendez-vous and of contents increases the number of necessary iterations to guarantee that the content is located in the same requester’s sub-group.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish Research Council under project TEC2005-08068-C04-01/TCM and with funds of DG Technological Innovation and Information Society of Industry and Environment Council of the Regional Government of Murcia and with funds ERDF of the European Union.

REFERENCES


Keywords: Scheduling, Multi-Agent Systems, Supply-Chain Management.

Abstract: The approach to scheduling presented in this article is applicable to multi-agent cooperative supply-chain production-distribution scheduling problems. The approach emphasises a scheduling temporal perspective, it is based on a set of three steps each agent must perform, in which the agents communicate through an interaction protocol, and presupposes the sharing of some specific temporal information (among other) about the scheduling problem, for coordination. It allows the set of agents involved to conclude if a given scheduling problem has, or has not, any feasible solutions. In the first case, agent actions are prescribed to re-schedule, and so repair, a first solution, if it contains constraint violations. The resulting overall agent scheduling behaviour is cooperative. We also include some results of the application of the approach based on simulations.

1 INTRODUCTION

In this article we present an approach to scheduling in cooperative supply-chain production-distribution scheduling environments, including some unpublished details of the same work.

Scheduling is the allocation of resources over time to perform a collection of tasks, subject to temporal and resource capacity constraints (Baker 1974). For classical, Operations Research (OR) based, approaches to scheduling see (Blazewicz 1994); for more modern approaches, Artificial Intelligence (AI) based, see (Zweben 1994), for instance. Planning and coordination of logistics activities (production, distribution) has been the subject of investigation since around 1960, in the areas of OR/Management Science (Graves 1993). More recently, some attention has been paid to scheduling in this kind of environments (e.g., see (Kjenstad 1998) or (Rabelo 1998)).

In our work, the specific logistics context of cooperative supply-chain/Extended Enterprise (EE) (O'Neill 1996) is considered. The EE is usually assumed to be a kind of Virtual Organisation, or Virtual Enterprise, where the set of participant agents (enterprises) is relatively stable (for concepts and terminology see pages 3-14 in (Camarinha-Matos 1999); in this last work, other approaches to scheduling in this kind of context can be found).

The main features of the scheduling problem are: a) decision is decentralised and distributed among multiple autonomous agents, b) problem solution involves communication and cooperation among agents, and c) scheduling can be highly dynamic.

For the modelling of the environment we adopt the AI Multi-Agent Systems paradigm (O'Hare 1996), and consider a network of agents linked through client-supplier relationships and communication channels. Capacity, or manager, agents, manage, each one, the limited capacity of an individual resource, specialised in either production or transportation or store tasks, the last ones with flexible durations. Producer and transporter agents are both termed processors, as their capacity is based on a product rate; store agent capacity is based on a product quantity. A supervision agent introduces work in the system, and fictitious retail and raw-material agents define the frontiers of the network with the outside at the downstream and upstream extremes, respectively.
Our temporal scheduling approach is described in (Reis 2001a) for processor agents, (Reis 2001b)] includes a three-step procedure and processor agent re-scheduling cases, and (Reis 2001c) includes store agent re-scheduling cases (our earlier work is referred in these articles). Here we include also some demonstration examples, taken from (Reis 2002), which exposes our whole model and approach. The following sections present: the high level agent interaction protocol used, basic concepts underlying the approach, the steps of the approach, some demonstration examples, and a conclusion.

2 INTERACTION PROTOCOL

In Figure 1 we present the high level agent interaction protocol used by capacity agents, defined through a pair of symmetrical conversation models (Request-from-Client and Request-to-Supplier, shown as state diagrams) in the context of which certain types of messages (also shown and described) can be exchanged.

In Figure 2 we show an example of a network job built by agents of a hypothetical agent network for a scheduling problem. The precedence relationships among the tasks (the arrows forming a tree) reflect the client-supplier relationships among the agents.

A scheduling problem is introduced by the network supervision agent \( g_{\text{RT}} \), through a global interval \( H = \langle \text{RD}, \text{DD} \rangle \) (where \( \text{DD} \) and \( \text{RD} \) are the global hard temporal limits), and a global request from outside \( \text{d} \), containing retail agent identification, product, quantity and date for satisfaction (the request due-date, \( \text{dd} = \text{TIME}(\text{d}) \)). In forming the job depicted in Figure 2, retail agent \( g_{\text{RT}} \) first receives from \( g_0 \) values of \( \text{DD} \) and \( \text{d} \), then sends a request type message to capacity agent \( g_1 \), essentially containing \( \text{d} \). Starting from \( g_1 \), agents in the client-supplier tree then perform a set of communicative actions (sending request messages containing local requests to one or more suppliers to ask for task supplies). This upstream propagation of local requests ends with the raw-material agents \( g_{\text{R1}}, g_{\text{R2}} \) and \( g_{\text{R3}} \) passing to \( g_0 \) the local requests of capacity agents \( g_3, g_4, \) and \( g_5 \), as global requests to outside. Subsequently, these
agents receive from \( g_0 \) the value of \( RD \), and then, acceptance messages are propagated downstream, starting from the raw-material agents. For each of the capacity agents, an acceptance message confirms its task, which is then scheduled. According to the approach we propose (see ahead), if any agent in the client-supplier tree detects that the problem has no feasible solution, or receives a rejection message from a supplier, it sends a rejection message to its client and cancels the accepted requests of its suppliers. This would lead to failure in establishing the job, with rejection of the scheduling problem by the agent network as a whole.

After the establishment of a job for a scheduling problem, re-request, re-acceptance and re-rejection messages can be used by the agents to ask for, accept or reject re-scheduling requests to, or from, its client or suppliers, to repair the initial solution schedule, in the case they locally detect temporal or capacity constraint violations. As a last choice, agents can resort to cancellation messages, if a feasible solution cannot be found. This can happen because, as the environment is dynamic, new scheduling problems appear and the individual agent resource capacities are limited.

### 3 COOPERATIVE APPROACH

The approach we propose is similar to some others that operate through scheduling by repair (a first, possibly non-feasible, solution is found which is then repaired through search, if necessary; see (Minton 1992), for instance). It is based on a set of three steps performed by each individual capacity agent for each scheduling problem involving the agent (specifically, involving an agent client-supplier tree that includes the agent), occurring after the problem is known by the agent, i.e., after receiving the respective client request message. In the first two steps the approach emphasises scheduling from a temporal perspective and results in an overall agent cooperative scheduling behaviour.

In Figure 3 a set of temporal parameters used in the approach are represented along timelines for a processor agent and for a store agent; processor \( g_j \) and store \( g_l \), involved in the job depicted in Figure 2, are used as an example. Besides the agent task (labelled \( O \)) and the requests from the client and to the supplier(s) (labelled \( d \)), a set of temporal intervals (labelled \( h \) and \( H \)) and slacks (labelled \( FJ, FEJ, fi, fim, FEM \) and \( FM \)) is shown. These are defined in the following. In the case of \( g_7 \), two suppliers are needed so, there are two \( h \) and two \( H \) intervals:

\[
\begin{align*}
h_{i,14}^7 &= \text{TIME}(d_{i,14}^7), \text{TIME}(d_{i,14}^4) > \\
H_{i,14}^7 &= \text{RD}_{i,14}^7, \text{DD}_{i,14}^7 > \\
& (j=8, 9)
\end{align*}
\]

where \( DD_{i,14}^7 \) is the local hard temporal limit for the end time of task \( O_{i,14}^7 \) (i.e., considering all agent tasks downstream \( g_j \), scheduled as late as possible), and \( RD_{i,14}^7 \) is the local hard temporal limit for the start time of the \( O_{i,14}^7 \) concerning to supplier \( g_j \), starting on supplier \( g_k \), scheduled as early as possible); in determining these local temporal limits, a minimum duration of 1 time unit for store tasks (which have flexible duration) is considered.

For \( g_j \) internal downstream and upstream slacks:

\[
\begin{align*}
fi_{j,14}^7 &= \text{TIME}(d_{i,14}^4), \text{END}(O_{i,14}^7) \\
fi_{i,14}^7 &= \text{START}(O_{i,14}^7), \text{TIDE}(d_{i,14}^4) \\
& (j=8, 9)
\end{align*}
\]

For \( g_j \) external downstream and upstream slacks:
FEJ, = DD, - TIME (d, )
FEM, = TIME (d, ) - RD, 

For g, downstream and upstream slacks:
FJ, = FEJ, + fi, 
FM, = FEM, + fim, 

For g, intervals (stores have one h, and one H):
h, = < TIME (d, ), TIME (d, ) >
H, = < RD, , DD, >

(time unit minimum duration for the task and the rest of the effective duration considered as internal slack, being:
fi, + fim, = DURATION (O, ) - 1

Additionally, for any g, we define the total slack:
FK, = FJ, + FM, 

where FM, = FM, , using for FM, the upstream slack corresponding to the most restrictive H, , for processors with more than one supplier.

Assuming agents always maintain non negative internal (fi, and fim,) slacks and, in the case of store agents, the minimum task duration is 1 time unit, for temporal constraints to be respected, the following conditions must hold. For processor g:
DURING (INTERVAL (O, ), h, ) 

DURING (h, , H, )

Similar conditions must hold for store g. The conditions mean that, for an agent scheduling problem, an O interval must be contained in the h interval(s), and each h interval must be contained in the corresponding (same supplier) H interval. This is equivalent to say that all values for slacks FJ, FM, FEJ and FEM must be non negative. If any of the conditions described doesn’t hold, an agent must engage in a re-scheduling activity, involving communicative actions to agree on acceptable temporal values of requests with the client or the supplier(s) and, possibly, re-scheduling actions to correct the temporal position of the task interval (which must be, at least, inside the H interval).

However, before engaging in such activity, an agent must be sure that the problem is time-feasible (otherwise it must be rejected), i.e., that the most restrictive H interval duration is greater than or equal to the task duration (using for stores a minimum of 1). This is ensured if, for any agent g:
FK, ≥ 0

In order to be able to determine the values for the end-points of H intervals, an agent receives from the client (via request message) the value of the FEJ slack; then, ensuring non negative internal slack values for the task to be scheduled, it will send to each supplier the supplier FEJ value (via request messages); the agent FEM slack values are received from the suppliers (via acceptance messages), if they accept the requests; in the case the problem is time-feasible, the agent finally schedules its task and sends to the client the client FEM value (via acceptance message). For instance, for agent g, the H’s end-points are given by
DD, = TIME (d, ) + FEJ, and RD, = TIME (d, ) - FEM, (j = 8, 9); supplier g FEJ value is given by FJ, + fim, , and client FEM value by MIN (FM, ) + fim, ; retail agent g passes the value of DD-TIME (d) to its supplier capacity agent g, as g FEJ value, and each of the raw-material agents g (m = 17, 18, 19) passes the value of TIME (d, ) - RD to its client capacity agent g (k = 8, 11, 12), as g FEM value.

4 STEPS OF THE APPROACH

The approach we propose is a minimal approach, i.e., agents will only modify a scheduling problem solution if it contains constraint violations and, in that case, they operate minimal re-scheduling corrections. The approach is composed of the following sequence of three agent steps:

Step 1. Acceptance and initial solution - If any request to a supplier was rejected, reject the request from the client, cancel the accepted requests to suppliers, and terminate the procedure (with failure). Otherwise, see if the problem is temporally over-constrained; if it is, terminate the procedure
solution - If the established solution is time-feasible, proceed to Step 3. Otherwise, re-schedule requests, or requests and task, to remove all temporal constraint violations;

Step 3: Re-schedule to find a feasible solution - If the solution is resource-feasible (i.e., it has no capacity constraint violation), terminate the procedure (with success). Otherwise, try to re-schedule to remove all capacity constraint violations, without violating temporal constraints; if this is possible terminate (with success). As a last choice, resort to cancellation, together with task un-scheduling (terminating with failure).

Figure 4: Examples of Step 2 re-scheduling cases 1, 2, 3 and 4, for a processor agent, with situations before, and after, minimal re-scheduling actions.

(with failure) by rejecting the problem, i.e., reject the request from the client and cancel all accepted requests to suppliers; if it isn’t, establish an initial solution and proceed to Step 2;

Step 2: Re-schedule to find a time-feasible

Figure 5: Examples of Step 2 re-scheduling cases 1, 2, 3 and 4, for a store agent, with situations before, and after, minimal re-scheduling actions. In order to detect cases 1 and 3, the minimum duration task interval is considered shifted to the extreme left, and for cases 2 and 4 shifted to the extreme right, relatively to the effective task interval.
For a time-feasible scheduling problem, this procedure results in the agents of the client-supplier tree building first, an initial, possibly flawed, solution (in Step 1), which can then be repaired (in Step 2), if necessary. Steps 1 and 2 are oriented to a temporal perspective and concern only to a single problem of an individual agent; Step 3 is oriented to a resource perspective and involves all problems of the agent at Step 3, as all the tasks of the agent compete for its resource capacity.

Table 3 Step 2: Re-scheduling for a Time-Feasible Solution

<table>
<thead>
<tr>
<th>dd = 25</th>
<th>Time-Feasible Solution</th>
<th>1155</th>
</tr>
</thead>
<tbody>
<tr>
<td>H = 4.22</td>
<td>Messages exchanged among agents</td>
<td></td>
</tr>
<tr>
<td>g4 g14</td>
<td>g1</td>
<td>g4</td>
</tr>
<tr>
<td>g1 g4</td>
<td>g4</td>
<td>dd = 24</td>
</tr>
<tr>
<td>g4 g7</td>
<td>g7</td>
<td>dd = 20</td>
</tr>
<tr>
<td>g7 g8</td>
<td>g8</td>
<td>dd = 13</td>
</tr>
<tr>
<td>g7 g9</td>
<td>g9</td>
<td>dd = 11</td>
</tr>
<tr>
<td>g9 g11</td>
<td>g11</td>
<td>dd = 6</td>
</tr>
<tr>
<td>g9 g12</td>
<td>g12</td>
<td>dd = 5</td>
</tr>
<tr>
<td>request g12 g19</td>
<td>g19</td>
<td>dd = 2</td>
</tr>
<tr>
<td>acceptance g17 g8</td>
<td>g8</td>
<td>FJ / JM = 5</td>
</tr>
<tr>
<td>acceptance g18 g11</td>
<td>g11</td>
<td>FJ / JM = -3</td>
</tr>
<tr>
<td>acceptance g19 g12</td>
<td>g12</td>
<td>FJ / JM = -2</td>
</tr>
<tr>
<td>acceptance g11 g9</td>
<td>g9</td>
<td>FJ / JM = -1</td>
</tr>
<tr>
<td>acceptance g12 g9</td>
<td>g9</td>
<td>FJ / JM = 0</td>
</tr>
<tr>
<td>acceptance g8 g7</td>
<td>g7</td>
<td>FJ / JM = 7</td>
</tr>
<tr>
<td>acceptance g9 g7</td>
<td>g7</td>
<td>FJ / JM = 1</td>
</tr>
<tr>
<td>acceptance g7 g4</td>
<td>g4</td>
<td>FJ / JM = 7</td>
</tr>
<tr>
<td>acceptance g4 g1</td>
<td>g1</td>
<td>FJ / JM = 9</td>
</tr>
<tr>
<td>acceptance g1 g14</td>
<td>g14</td>
<td>FJ / JM = 9</td>
</tr>
</tbody>
</table>

b) Schedule data after Step 2. Figure 8: Scheduling problem 1155 Step 2 data. Some requests were re-scheduled to remove temporal constraint violations (the negative slacks in Figure 7-c).

5 EXAMPLES

We now present two network scheduling problem simulation cases, together with the results of the application of the three-step procedure described, assuming the job depicted in Figure 2 (see Figure 6 for the first problem, and Figure 7, Figure 8 and Figure 9 for the second). As input data for problem simulation, global temporal parameters (RD and DD of global H interval and date value dd=TIME (d) of the global request from outside), as parameters for agent g0, and task durations (d) and initial values for internal slacks (fi j and fim, which can be further changed by agents) for each capacity agent are given.

Figure 6 shows the initial data (a), and the messages exchanged (b) and resulting schedule data (c) in Step 1, for the first problem (labelled problem...
a) Schedule resulting from Step 1 (built from data in Figure 7-c). Concerning to temporal constraint violation situations experienced by the agents, as shown, agents $g_1$ and $g_4$ have a case 1 situation, agent $g_7$ has a case 3 situation, agent $g_9$ has a case 4 situation and agents $g_{11}$ and $g_{12}$ have a case 2 situation.

Figure 7 shows the initial data (a), and the messages exchanged (b) and resulting schedule data (c) in Step 1, for the second problem (labelled problem P 1155). As Figure 7-c shows, no capacity agent detected a negative value for total slack $FT$, so the problem is not temporally over-constrained.

As a result, no rejection or cancellation messages are exchanged until the end of Step 1, see Figure 7-b. The resulting network schedule in Step 1 is shown in Figure 9-a. In Step 2, temporal constraint violation situations of case 1 for agents $g_1$ and $g_4$, of case 3 for agent $g_7$, of case 4 for agent $g_9$, and of case 2 for agents $g_{11}$ and $g_{12}$ are detected. Solution repair is accomplished by agents through inter-agent local request re-scheduling (for all those agents), and additional agent task re-scheduling (only for agents $g_1$, $g_4$, $g_{11}$ and $g_{12}$), according to the minimal actions prescribed. Figure 8 shows the messages exchanged (a) and resulting schedule data (b), and Figure 9-b shows the resulting schedule in Step 2, for this problem. As shown by Figure 9-b, all temporal constraint violations found in the initial solution (Figure 9-a) disappeared.
6 CONCLUSION

We described an approach to multi-agent scheduling in a cooperative supply-chain environment. The approach presupposes the use of an agent interaction protocol (also described), is based on a three-step procedure prescribed for each agent involved in a scheduling problem, and results in an individual cooperative scheduling behaviour. In Step 1 agents detect if the problem is temporally over-constrained and, if it's not, they schedule an initial, possibly non-time-feasible, solution (otherwise, they reject the problem). The exchange of specific temporal slack values, besides product, quantity and due-date information, used as a scheduling coordination mechanism, allows the agents to locally perceive the hard global temporal constraints of the problem, and rule out non-time-feasible solutions in the subsequent steps. Each of these pieces of information exchanged in Step 1 corresponds, for a particular agent, to a sum of slacks downstream and upstream the agent in the agent network, and cannot be considered private information of any agent in particular. If necessary, in Step 2, agents repair the initial solution, through re-scheduling, in order to obtain a time-feasible one. In Step 3 any capacity constraint violation must be removed, either through re-scheduling, or by giving up the problem.

No specific details were given for Step 3. In fact, this is the matter of our current and future work. Step 3 can be refined to accommodate additional coordination mechanisms for implementing certain solution search strategies. For instance, strategies based on capacity/resource constrainedness (see [Sycara 1991] or [Sadeh 1994]), to lead the agents on a fast convergence to both time and capacity-feasible solutions, including solutions satisfying some scheduling preferences, or optimising some criteria, either from an individual agent perspective, or from the global perspective of the overall system.

REFERENCES

Reis 2001c. Reis, J.; Mamede, N., Scheduling, Re-Scheduling and Communication in the Multi-Agent Extended Enterprise Environment, accepted for the MASTA’01 Workshop, EPIA’01 Conference, December, 17-20, 2001, Porto, Portugal.
TOWARDS A QUALITY MODEL FOR GRID PORTALS

Mª Ángeles Moraga, Coral Calero, Mario Piattini
Alarcos Research Group. UCLM-SOLUZIONA Research and Development Institute. University of Castilla-La Mancha
{MariaAngeles.Moraga, Coral.Calero, Mario.Piattini}@uclm.es

David Walker
School of Computer Science. Cardiff University
David.W.Walker@cs.cardiff.ac.uk

Keywords: Quality models, Grid portals.

Abstract: Researchers require multiple computing resources when conducting their computational research; this makes necessary the use of distributed resources. In response to the need for dependable, consistent and pervasive access to distributed resources, the Grid came into existence. Grid portals subsequently appeared with the aim of facilitating the use and management of distributed resources. Nowadays, many Grid portals can be found. In addition, users can change from one Grid portal to another with only a click of a mouse. So, it is very important that users regularly return to the same Grid portal, since otherwise the Grid portal might disappear. However, the only mechanism that makes users return is high quality. Therefore, in this paper and with all the above considerations in mind, we have developed a Grid portal quality model from an existing portal quality model, namely, PQM. In addition, the model produced has been applied to two specific Grid portals.

1 INTRODUCTION

Nowadays, many users have access to, and require, multiple computing resources to conduct their computational research (Dahan et al., 2004). This makes the use of distributed resources necessary. For this reason and with the aim of providing dependable, consistent and pervasive access to distributed resources, the Grid emerged (Li et al., 2003). The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations (Foster et al., 2001).

Specifically, the Grid couples a wide variety of geographically distributed resources such as PCs, workstations and clusters, storage systems, data sources, databases and special purpose scientific instruments and presents them as a unified, integrated resource (Li et al., 2003).

The main problem with the Grid, however, is the difficulty involved in using grid resources. That is due to its complex architecture. Therefore, in order for scientists to use grid resources effectively as a problem solving infrastructure, transparent and easy-of-use interfaces to the complex set of grid resources are necessary (He and Xu, 2003). Nowadays, Grid Ports are coming into existence to resolve this problem. They can be considered as a mechanism for providing user-friendly access to grid resources, and consistent access patterns, as well as easy usage of grid services. The original objective of this portal type was to create web-accessible problem-solving environments (PSEs) that allowed scientists to access distributed resources, and to monitor and execute distributed Grid applications from a Web browser (Lin and Walker, 2004). Although at the beginning these portals were aimed at researchers, nowadays they can be used by any user who wants to use distributed resources.

Many Grid portals exist at the present time. An immediate effect of this widespread presence is the increasing range of resources available at the click of a mouse, that is, without the user wasting time and money by physically moving from one place to another (Cox and Dale, 2001; Singh, 2002). It is because of this that portals must offer a good level of quality, thus users are attracted to them and come back regularly.

Bearing this in mind, as well as the lack of quality models specifically for Grid portals, in this
paper we present a Grid portal quality model (G-PQM) created from an existing portal quality model, namely, PQM (Portal Quality Model) (Moraga et al., 2004b).

The rest of the paper is organised as follows. In section 2 the quality model for Grid portals is shown while in section 3 this quality model is applied to two Grid portals. Finally, section 4 concludes and outlines further work.

2 QUALITY MODEL FOR GRID PORTALS

Grid portals appeared because of the need to make access by researchers to Grid resources easier. The developers of Grid portals seek to ensure that users return to their portal often. However, the only mechanism that makes users return is high quality (Offutt, 2002). Therefore, a quality model which is specifically for Grid portals, namely G-PQM (Grid Portal Quality Model), has been developed. The usefulness of this model is two-fold. On the one hand, this model helps users to evaluate the different Grid portals and to choose the one with the highest quality. And on the other hand, the model’s dimensions can be used as indicators to help developers when building the portal.

To develop G-PQM a quality model for web portals, namely PQM (Portal Quality Model), was used as the basis. PQM is composed of six dimensions and seeks to determine the strong and weak points of a specific portal. We can also define corrective actions for the weaknesses, and improve the quality level of a portal (Moraga et al., 2004a). In order to adapt this model to Grid portals, some definitions of the dimensions have been modified and, additionally, some dimensions have been inserted. In Figure 2, we can see the different phases used in developing the Grid portal quality model, G-PQM.

In our introduction, the first phase “Study of the Grid portals context” was presented.

2.1 Adaptation of the PQM Dimensions

We have adapted the following PQM dimensions:
- **Tangible**: This dimension indicates if “the Grid portal contains all the software and hardware infrastructures needed according to its functionality”.
  - Adaptability: ability of the Grid portal to be adapted to different devices (for instance, PDA, PCs, mobile phone, etc.).
  - Transparent access: ability of the Grid portal to provide access to the Grid resources while isolating the user from their complexity.
- **Reliability**: It is the “ability of the portal to perform the specified services”. In addition, this dimension will be affected by:
  - Fault tolerance: capability of the Grid portal to maintain a specified level of performance in the event of software faults (ISO, 2001) (for example, a fault during the sending or the execution of a job).
  - Availability: The portal must be always operative in order for users to be able to access it and use its Grid resources anywhere and anytime.
  - Search Quality: The results that the portal provides when undertaking a search must be appropriate to the request made by the user.
  - Quality in the use of resources: the user can use Grid resources under specified conditions with the portal.

![Figure 1: Phases for the construction of the G-PQM model.](image-url)
• **Responsiveness**: It is the “willingness of the Grid portal to help and to provide its functionality in an immediate form to the users”. In this dimension, we note the following sub-dimensions:
  o **Scalability**: This refers to the ability of the portal to adapt smoothly to increasing workloads coming about as a result of additional users, an increase in traffic volume or the execution of more complex transactions (Gurugé, 2003).
  o **Efficient access**: This relates to the response times experienced by portal users (Gurugé, 2003).

• **Empathy**: We define this dimension as the “ability of the Grid portal to provide caring and individual attention”. In this dimension, we observe the following sub-dimensions:
  o **Navigation**: The Grid portal must provide simple and intuitive navigation when being used.
  o **Presentation**: The Grid portal must have a clear and uniform interface.
  o **Integration**: All the components of the Grid portal must be integrated in a coherent form.
  o **Personalization**: The portal must be capable of adapting to the user’s priorities.

• **Data and information files quality**: This dimension is defined as the “quality of the data contained in the portal and of the files which specify the available services in the portal and the names of devices responsible for these services”. According to Dedeke and Kahn, we can distinguish four different subdimensions (Dedeke and Kahn, 2002):
  o **Intrinsic**: this indicates what degree of care was taken in the creation and preparation of data/files.
  o **Representation**: this indicates what degree of care was taken in the presentation and organization of data/files for users.
  o **Contextual**: to what degree the data/files provided meet the needs of the users.
  o **Accessibility**: this indicates what degree of freedom users have to use data, define and/or refine the manner in which data/files are inputted, processed or presented to them.

2.2 **Inserting New Dimensions**

The following dimension has been added:
• **Security**: This is “the ability of the portal to prevent, reduce and properly respond to malicious harm” (Firesmith, 2004). This dimension will be affected by:
  o **Access control**: capability of the portal to allow access to its resources only to authorized persons. Thus, the portal must be able to identify, authenticate and authorize its users.
  o **Security control**: the capability of the Grid portal to carry out auditing of security and detect attacks. The auditing of security shows the degree to which security personnel are enabled to audit the status and use of security mechanisms by analyzing security-related events. In addition, attack detection seeks to detect, record and notify attempted attacks as well as successful attacks.
  o **Confidentiality**: Ability to maintain the privacy of the users.
  o **Integrity**: the capability of the portal to protect components (of data, hardware, and software) from intentional or unauthorized modifications.

2.3 **Definitive Model (G-PQM)**

Taking into account the dimensions which have been adapted as well as the dimensions that have been introduced, the following model results (Figure 3):
3 APPLYING G-PQM

Having defined G-PQM, the next step is to apply it to some Grid portals with the objective of determining, on the one hand, the extent to which these portals satisfy the dimensions identified in the Grid portal quality model; and on the other hand, to identify possible improvements in the quality of these portals.

In our first approach, G-PQM has been applied to two Grid portals. It should be noted that we have applied G-PQM from the point of view of the users. G-PQM is, however, directed at portal developers. For this reason, some of the identified dimensions or sub-dimensions may not be measured (in this case, we will assign the value “not evaluable” to the (sub) dimension). In spite of this, we can obtain an overall assessment of the quality of these Grid portals.

3.1 GridPort Demo Portal

As a first step, the model has been applied to the GridPort demo portal which is a fully operational test portal that is intended to serve as a starting point for those interested in grid portal development (the reader can find more information about this portal at http://gridport.net/main/). This portal has been developed using the GridPort toolkit which enables the rapid development of highly functional grid portals that simplify the use of underlying grid services for the end-user (GridPort, 2006). The GridPort demo portal includes portlets that allow a user to do the following: view static and dynamic information about the resources in a grid, obtain short-term proxies from a myproxy server, submit batch jobs to resources on the grid, and browse and transfer files between resources on the grid (GridPort, 2006).

The outcomes obtained are the following:

- Tangible:
  - Adaptability: The following software packages are prerequisites to using the GridPort Demo Portal: JDK 1.4.2, Jakarta Ant 1.6, TomCat, etc. These packages cannot be installed on all devices.
  - Transparent access: GridPort has Grid portlets whose aim is to provide transparent access to resources.
- Reliability:
  - Fault tolerance: Not evaluable.
  - Availability: During the testing, the portal was available anywhere and anytime.
  - Search Quality: Not applicable because the portal does not have a search engine.
- Quality in the use of resources: Not evaluable.
- Responsiveness:
  - Scalability: The portal is not limited to a specific number of users.
  - Efficient access: During the testing, the time between the request for a page and obtaining it was found to be acceptable.
- Security:
  - Access control: The portal has mechanisms to identify (asking for username and password) and authenticate (has GridSphere authentication modules) users. Moreover, it has the capacity to authorize certain users to use certain resources.
  - Security control: Not evaluable.
  - Confidentiality: Not evaluable.
  - Integrity: users cannot carry out unauthorized actions.
- Empathy:
  - Navigation: The navigation is simple and intuitive.
  - Presentation: The interface is clear and uniform.
  - Integration: All the components of the Grid portal appear in a coherent, integrated form.
  - Personalization: The portal can adapt to the user’s priorities.
- Data and information files quality:
  - Intrinsic:
    - From the point of view of data: Not evaluable.
    - From the point of view of information files: Not evaluable.
  - Representation:
    - From the point of view of data: During the testing, the data were presented in an organized form.
    - From the point of view of information files: Not evaluable.
  - Contextual:
    - From the point of view of data: the information obtained during the testing satisfied our needs.
    - From the point of view of information files: Not evaluable.
  - Accessibility:
    - From the point of view of data: users do not influence the manner in which data are inputted, processed or presented to them.
    - From the point of view of information files: Not evaluable.

We must take into account the fact that we have carried out the assessment from the point of view of the end user. That being so, we do not have all the necessary data, so the conclusions obtained from applying G-PQM are not as definitive as they should be. However, we can see that the main characteristics which must be improved are:
adaptability (because the number of minimum requirements is excessive and this makes it impossible to adapt the portal to an arbitrary device) and data accessibility (because users cannot influence the way in which data are inputted, processed or presented to them). The rest of the characteristics which have been assessed, have given a favourable result. It would likewise be interesting to obtain more information related to the portal, for the purpose of detecting other weak points. We could thereby improve portal quality.

3.2 OGCE Portal

Secondly, we have applied the model to the OGCE portal, whose objective is to create an environment that facilitates the use of Grid resources. The results obtained from applying G-PQM are:

- Tangible:
  - Adaptability: The minimum requirements are: 500 MB free hard-disk space, Pentium III or higher (or a similarly capable processor) and 128 MB free RAM.
  - Transparent access: OGCE Port (release 2) has Grid portlets which manage remote files, execute remote commands, etc. Furthermore, this portal has inter-portlet communication tools that allow portlets to share data.

- Reliability:
  - Fault tolerance: Not evaluable.
  - Availability: The portal was available anywhere and anytime.
  - Search Quality: Not applicable because the portal does not have a search engine.
  - Quality in the use of resources: Not evaluable.

- Responsiveness:
  - Scalability: The portal is not limited to a specific number of users.
  - Efficient access: The response time was very high in some testing, and the request was not even met in some instances.

- Security:
  - Access control: The portal has mechanisms to identify (asking for username and password) and authenticate (has GridSphere authentication modules) users. Moreover, it has the capacity to authorize certain users to use certain resources.
  - Security control: Not evaluable.
  - Confidentiality: Not evaluable.
  - Integrity: users cannot carry out unauthorized actions.

- Empathy:
  - Navigation: The navigation is simple and intuitive.

- Data and information files quality:
  - Intrinsic:
    - From the point of view of data: Not evaluable.
    - From the point of view of information files: Not evaluable.
  - Representation:
    - From the point of view of data: During the testing, the data were presented in an organized form.
    - From the point of view of information files: Not evaluable.
  - Contextual:
    - From the point of view of data: The information obtained during the testing satisfied our needs.
    - From the point of view of information files: Not evaluable.
  - Accessibility:
    - From the point of view of data: users do not influence the way in which data are inputted, processed or presented to them.
    - From the point of view of information files: Not evaluable.

As with the previous case, we have applied our model from the point of view of the end user, so there are some dimensions which cannot be assessed. However, taking into account the dimensions we have assessed, we can see that the following tasks to improve portal quality could be carried out: reduction of the number of minimum requirements, so as to allow the portal to adapt itself to any device; improvement of the efficiency of access; and above all, avoidance of a request not obtaining an answer and elimination of the appearance of a blank screen. On the other hand, we have obtained favourable results for the rest of the characteristics we have assessed. It will also be of interest to us to obtain information related to the dimensions which have not been assessed.

4 CONCLUSIONS AND FUTURE WORK

Nowadays, many scientists require the use of the Grid to conduct their computational research. However, its use is not a trivial task. For this reason, and with the aim of allowing an easy access to Grid
resources via a Web browser interface, Grid portals have come into existence.

Many different Grid portals can be found at the present time. Therefore, it is easy for users to move from one Grid portal to another, without the user wasting time and money. Thus, for users to be attracted to a particular Grid portal and come back regularly, the portal must offer a good level of quality.

Bearing all this in mind, a quality model for Grid portals, namely G-PQM, has been presented. This model can be used, on the one hand, to assess the quality level of a specific Grid portal, and on the other hand, to identify its weaknesses and define corrective actions which improve its level of quality. In addition, this model has been applied to two Grid portals and some corrective actions have been defined in order to improve their level of quality.

Future work includes the validation of the model characteristics through surveys. In addition, measures for each one of the characteristics and sub-characteristics must be identified. Thereby, the G-PQM will be finished.

ACKNOWLEDGEMENTS

This work was conducted when the first author was in stage at the University of Cardiff and is part of the CALIPO (TIC 2003-07804-C05-03) and DIMENSIONS (PBC-05-012-1) projects and the CALIPSO network (TIN2005-24055-E).

REFERENCES


1 INTRODUCTION

In the past, advocates of the service-oriented architecture (SOA) have predicted that the successful integration of loosely-coupled services belonging to different, sometimes competing, but always collaborating organizations, would storm the world. It would create a myriad of new business opportunities, enabling the formation of virtual organizations where SMEs would join forces to thrive in ever increasingly competitive global markets. Yet the industry has been slow to deploy its SOAs, and the degree of integration between different organizations remains low.

At the moment the developer faces a situation where the tools originally used to produce intra-organizational, non-distributed applications are already overstretched to cope with issues of distribution across organizational domains. Furthermore, collaboration presumes a minimum level of mutual trust, and wherever trust is not considered sufficient, businesspeople turn to contracts as a mechanism to reduce risks. In other terms, for the SOA to deliver its promised advantages, developers need not only language support for distribution, but also cost effective contract management solutions. Researchers and industries alike have begun addressing this very essential issue with a top-down approach. Several electronic contract languages, their models and reasoning techniques are in the process of being discussed and refined. Thus we see a pressing need to provide the actual system developers with the means to implement services that meet the requirements dictated by such contracts.

In the next section, we recall the main features of SOAs and discuss the requirements posed by contracts. In Sec. 3, we discuss programming languages and SOA. In Sec. 4, we identify open problems and in Sec. 5 we discuss possible concrete scenarios for addressing them.

2 SOA AND CONTRACTS

In a SOA, applications are essentially distributed systems composed of services (Fig. 1, borrowed from (Papazoglou, 2003)). A service is a loosely-coupled, technology neutral and self-describing computation element. Loose coupling is achieved through encapsulation and communication through message passing; technology neutrality results from adopting standardized mechanisms; and rich interface languages permit the service to export sufficient information so that eventual clients can discover and connect to it (Papazoglou, 2003). A SOA can be implemented in many different ways, e.g. using web services.
Web services exchange SOAP messages over standard Internet protocols which carry a payload built from a stack of open XML standards (WSA, 2004). There are strong similarities between services and components in a component-based system (Szyperski, 2003). However, services usually have a coarser granularity and the communication medium with its high latency and openness constrains reliability and security in ways that easily go beyond what can be found in most component-based systems.

The services in a SOA usually belong to different organizational domains and therefore there is no single line of authority regulating their interactions. In principle a consumer must trust the provider to deliver the expected service, or establish a contract with it. For our purposes, a contract describes an agreement between distinct services that determines rights and obligations on its signatories, and for which there exists static or dynamic ways of identifying contract violations.

In the case of a bilateral contract, one usually talks about the roles of service provider and service consumer; but multi-lateral contracts are also possible where the players may play other roles. A service provider may also use a contract template (i.e. a yet-to-be-negotiated contract) to publish the services it is willing to provide. As a service specification, a contract may describe many different aspects of a service, including functional properties and also non-functional properties like security, quality of service (QoS) and reputation.

**Contract Models** There exists a number of contract models for services. The business process standard ebXML describes a Collaboration Protocol Agreement as a contract between business partners that specifies the behavior of each service (by simply stating its role) and how information exchanges are to be encoded. IBM’s Web Service Level Agreement is an XML specification of performance constraints associated with the provision of a web service. It defines the sources of monitoring data, a set of metrics (i.e. functions) to be evaluated on the data, and obligations on the signatories to maintain the metric values within certain ranges. The set of predefined metrics and the structure of WSLA contracts are designed for services involving job submissions in a grid computing environment. The later WS-Agreement, a Global Grid Forum recommendation that has not reached the standard status yet, is based on WSLA, but adapted to more recent web-services standards, e.g. WS-Addressing and WS-Resource Framework. WS-Agreement is also parametric on the language used to specify the metrics.

A number of problems have previously been identified for these standards and specifications: They are restricted to bilateral contracts, lack formal semantics (and therefore it is difficult to reason about them), their treatment of functional behavior is rather limited and the sub-languages used to specify QoS and security constraints are usually limited to small application-specific domains. In order to remedy the situation researchers have produced contract taxonomies (Aagedal, 2001; Beugnart et al., 1999; Tòsic, 2005), formalizations using logics, e.g. classical (Davulcu et al., 2004), modal (Daskalopulu and Maibaum, 2001), deontic (Paschke et al., 2005b) and defeasible logic (Governatori, 2005), and formalizations based on models of computation – e.g. finite state machines (Molina-Jimenez et al, 2004) and Petri nets (Daskalopulu, 2000). The diversity of contract types, their applications and properties poses a serious challenge to the definition of a generic contract model. This, however, has been identified as a major precondition for the advancement of the area (Bouysou and Sifakis, 2005).

**Discovery and Negotiation** In a setup for contract-enhanced service provision, providers are expected to make service descriptions available for consumers to discover and choose among them. The description takes the form of a proto-contract, or template, setting the basis for negotiating the provision of the service. Specifications like ebXML and WS-Agreement define sub-languages for such contract templates, though they are usually attached to a very specific negotiation model. There is, however, a large body of research on contract negotiation protocols under different threat models, particularly in the area of agent-based systems (Andreoli and Castellani, 2001; Picard, 2003; Kraus, 2001).

**Monitoring** Monitoring presents an important list of challenges. First, monitoring data (including execution events and samplings of continuous processes) needs to be collected in a timely, reliable and trustworthy manner even within a distributed system. Moreover, monitors are usually weaved into the application code by specialists (not by ordinary program-
mers), creating complex dependencies that seriously affect the software development process.

Quality of Service According to the ARTIST roadmap (Bouyssounouse and Sifakis, 2005), quality of service is a “function mapping a given system instance with its full behavior onto some [quantitative] scale”. Typical QoS measures for web services include average response time, minimum communication bandwidth and peak CPU usage. QoS measures usually depend on the behavior of the environment as well as of the service, thus models tend to have a stochastic nature, although this is not really necessary for monitoring purposes.

Typically, contract languages for QoS of web services consist of three main sub-languages. Their purpose is to specify: (1) The QoS measures (i.e. functions) including their domains; (2) a mapping between elements in the execution model (e.g. observable events) and the domains of QoS measures; and (3) the constraints on QoS measurements (i.e. the obligations). The design of such languages is therefore centered around the concept of QoS measure function. However, realistic contracts are not easily modeled as a set of functions: they are built upon the fundamental concept of obligation, to which other concepts (like QoS measures) become accessory. For instance, the fulfillment or violation of an obligation may trigger other obligations. Function-based approaches need then to encode obligation performances as QoS measures. Moreover, the inclusion of time considerations becomes unnecessarily complicated.

Security So far, contract languages for security pay almost exclusive attention to access control issues – e.g. (de Win et al, 2005)– but it is evident that they should be enhanced to cover other areas of security such as integrity and confidentiality.

3 SOA AND LANGUAGES

Current programming language abstractions are not adequate for SOA, much less for web-service development. The industry develops web services using the object-oriented programming (OOP) paradigm which maps badly to document-based communication, i.e. SOAP-transported XML documents, required by web-services (Meijer et al., 2003). Besides, many current production OOP languages (e.g. Java and C#) are based on the shared-state model so they do not handle concurrency and message passing particularly well. Another criticism of OOP concerns reusability. Object-orientation provides two distinct mechanisms for composing concerns, namely aggregation and inheritance, which may be difficult to combine with synchronization, history information or multiple views. Thus OOP needs better abstraction mechanisms.

The programming language community has long identified the need to provide easier ways to extend the abstraction mechanisms of a language. One of the main approaches is that of aspect-oriented programming (AOP) (Filman et al., 2005), which helps separate cross-cutting concerns (like access control) from the main business logic. AOP is composed of a set of techniques, including code instrumentation and runtime interceptors.

A similar approach uses composition filters (CF) (Akşit et al., 1992), where the idea is not to replace the programming paradigm but to enhance the expressive power and maintainability of current object-oriented (OO) languages. CF may be considered a modular extension to the OO model with interface layers including the so-called filters. Advantages of CFs with respect to aspects are exposed in Bergmans and Akşit, 2001.

An alternative approach defines new kinds of languages that adapt themselves better to the challenges posed by web services. Some concentrate on bridging the gap between the program language and the XML objects that web services should exchange (Florescu et al., 2002), others provide abstractions to manipulate interfaces (Cooney et al., 2005), and others address asynchronous communication by means of message passing – e.g., C_ (Bierman et al., 2005). The latter combines features from two other research languages: (a) Polyphonic C# (Benton et al., 2004): a control flow extension with asynchronous wide-area concurrency, and (b) Xen (Meijer et al., 2003): a data type extension for processing XML and table manipulation. Along the same lines, the Xtatic project (Gapeyev and Pierce, 2003) aims at extending C# with native XML processing adding regular types and regular patterns. More oriented to web service development, a new language is proposed in (Cooney et al., 2005) which combines XQuery’s semantics with imperative constructs and a join calculus-style concurrency model. This proposal solves some of the problems of mainstream languages (e.g., Java and C#) like concurrency and message correlation problems. It lacks, however, useful features like interface inheritance, correlated messages and garbage collection. Furthermore, the current implementation assumes a shared-state concurrency model.

Another interesting language is Creol (Johnsen and Owe, 2004), whose programs consist of concurrent objects with internal process control communicating asynchronously. By means of mechanisms for conditional processor release points, passive waiting, and time-out, explicit synchronization primitives are not needed in the language. Compared to for instance Polyphonic C#, Creol has a simpler set of communi-
cation primitives, using the concept of asynchronous method call, while maintaining multiple inheritance. It also offers a synchronized merge operator which effectively reduces the problems related to the so-called inheritance anomaly (Matsuoka and Yonezawa, 1993), while allowing compositional reasoning.

The ideal language for SOA and web-services development seems to be one which combines the advantages of the above-mentioned languages. It should be based on asynchronous communications with facilities for providing good abstractions for manipulating interfaces, XML processing and web service development. The inclusion of regular types a la XDuce (Hosoya and Pierce, 2003) and CDuce (Benzaken et al., 2003) would be a plus.

**Programming and Contracts** The solutions mentioned so far still lack support for discovery, monitoring and management of contracts. Approaches like AOP and CF can potentially provide some help here (Becker and Geihs, 2001), but they fail to abstract low-level issues and basically leave too much freedom to the programmer (which leads to code maintenance and analysis issues). This is an issue for which no consolidated work has been completed yet. Instead, one should look for different bits and pieces, depending on the characteristics of the contract.

Although contracts could cover many other aspects, guarantees of the timely provision of services are commonplace in SOA. Despite of the current wide acceptance of AOP as a good paradigm for improving reusability and modularity, there is no convincing solution to the application of aspects to real-time systems. In some cases (Tsang et al., 2004), aspect-orientation seems to perform better than object-orientation when dealing with real-time specifications, regarding system properties such as testability and maintainability. On the other hand, in (Assayad et al., 2005), there is a formal framework for multi-threaded and multi-processor software synthesis using timing constraints, where it is shown that AOP is not suitable for such cases.

A new concept for real-time system development (ACCORD) is presented in (Tesanovic et al., 2004), combining component-based and aspect-oriented software development. ACCORD bridges the gap between modern software engineering methods – focused mainly on component models, interfaces and separation of concerns – and real-time design methods. The focus is primarily on a design methodology, but not on analysis and verification of real-time systems. It is not clear, either, how the methodology could be used to develop asynchronous open distributed systems.

Programs using real-time features are, in general, difficult to design and verify, even more when combined with an inheritance mechanism. Changing application requirements or real-time specifications in real-time OO languages may produce unnecessary redefinitions. This is called the real-time specification inheritance anomaly. To solve it, (Aksit et al., 1994) propose real-time composition filters. Similar ideas could be applied to support contract-based system design.

A contribution towards verifying properties of contracts involving real-time as formulated in existing languages is found in (Diaz et al., 2005). They use a translation to a real-time model checker to verify the cooperation aspect of contracts.

In conclusion, there is still plenty of work to do in directly supporting development of services that can be trusted to implement their contracts.

### 4 RESEARCH DIRECTIONS

The main problems and open issues identified for supporting web services development include:

- Formal definition of generic contracts, in particular for QoS and security.
- Negotiable and monitorable contracts. Contracts must be negotiated till both parts agree on their final form, and they must be monitorable in the sense that there must be a way to detect violations. No existing programming language supports negotiable and monitorable contracts.
- Combination of object-orientation and concurrency models based on asynchronous message-passing. The shared-state based concurrency model is not suitable for web service development.
- Integration of XML into a host language, reducing the distance between XML and object data-models.
- Harmonious coexistence at the language level of real-time and inheritance mechanisms.
- Verification of contract properties. The integration of contracts in a programming language should be accompanied by good support for guaranteeing essential properties. Guaranteeing the non-violation of contracts might be done in (at least) four different ways: 1. with runtime enforcement, e.g. through monitors; 2. by construction, e.g. through low-level language mechanisms; 3. with standard static program analysis techniques; or 4. through model checking. None of the above can be used as a universal tool; they must be combined.

Addressing these issues and problems, we need to develop a model of contracts in a SOA that is broad enough to cater for at least contracts for QoS and security. A minimum requirement is the ability to seamlessly combine real-time models (for QoS specifications) and behavioral models (essential to constrain protocol implementation and to enforce security). Contract models should also address discovery
and negotiation. Yet, the formal definition of contracts should be only a first step towards a more ambitious task, namely to provide language-based support for programming and effective usage of such contracts. Some contracts may be seen as a wrapper which "envelopes" the code/object under the scope of the contract. Firewalls, for instance, may be seen as a kind of contract between the machine and the external applications wanting to run on that machine. It would be interesting to investigate a language primitive to create wrapped objects which are correct-by-construction. On the other hand, contracts for QoS and security could be modeled as first-class entities using a "behavioral" approach, through interfaces. In order to tackle timed constraints (related to QoS) such interfaces need also to incorporate time. As exposed in the ARTIST road-map, finding languages or notations for describing timing behaviors and timing requirements is easy; the real challenges are in analysis. Besides syntactic extensions, the language needs to have timing semantic extensions in order to allow extraction of a timed model, e.g. a timed automaton. This model may be checked with existing tools, e.g. Kronos (Yovine, 1997) and Uppaal (Bengtsson et al., 1995). Model checking tools will help to prove real-time properties, like guaranteeing that a service will satisfy its promised response-time constraints. Other properties may, instead, be proved to be correct-by-construction (e.g. wrappers).

In practice, many properties can only be proved correct through runtime approaches. A promising direction is to develop techniques for constructing runtime monitors from contracts, which will be used to enforce its non-violation.

5 FINAL DISCUSSION

The web is mostly used nowadays for retrieving remote information, but there is a high demand for more challenging applications that offer, negotiate and discover web services through XML interfaces. This new direction requires redesigning software architectures and revising the existing foundations of computer science (Montanari, 2004). Moreover, in order to make collaboration a reality among different web-servers, the formal definition of monitorable and negotiable contracts has become imperative. In this paper we have surveyed the main current approaches to programming in a SOA and their support in state-of-the-art programming languages. We have identified some problems and open issues of current approaches and we have proposed general research directions.

We believe object-orientation could still be a good paradigm for modeling open distributed systems, since its main problems come from sequential language design and implementation decisions, not from its original philosophy.

Regarding the formal definition of contracts, we believe such a generic model can be described harmoniously using real-time extensions of rewriting logic (Olveczky and Meseguer, 2002). This is in line with recent investigations in the use of rule languages to model contracts (Grosof and Poon, 2002; Paschke et al., 2005a). While these languages are essentially ad-hoc, we expect to profit from the existing large body of research in rewriting logics. The rule-based approach brings along new challenges in the definition of appropriate negotiation schemes (Reeves et al., 2001; Paschke et al., 2005c). Here again, rewriting logic can give invaluable help. Its reflection and meta-level computation properties may help define and structure the negotiation protocol.

Concerning the choice of a host language for incorporating contracts as first citizens, we believe Creol might be a good starting point. The Creol project has addressed many of the problems of OOP and it has a formal semantics defined in rewriting logic. More details on such a line of research can be found in the technical report (Giambiagi et al., 2006).

REFERENCES


Becker, C. and Geihs, K. (2001). Quality of Service and Object-Oriented Middleware-Multiple Concerns and their Separation. In ICDCSW.


A METHODOLOGY FOR ADAPTIVE RESOLUTION OF NUMERICAL PROBLEMS ON HETEROGENEOUS HIERARCHICAL CLUSTERS

Wahid Nasri, Sonia Mahjoub and Slim Bouguerra
ESSTT, 5 Avenue Taha Hussein - B.P. 56, Bab Menara - 1008 Tunis, Tunisia
Wahid.Nasri@ensi.rnu.tn

Keywords: Cluster computing, Adaptive techniques, Scheduling, Parallel algorithms, Matrix multiplication problem.

Abstract: Solving a target problem by using a single algorithm or writing portable programs that perform well is not always efficient on any parallel environment due to the increasing diversity of existing computational supports where new characteristics are influencing the execution of parallel applications. The inherent heterogeneity and the diversity of networks of such environments represent a great challenge to efficiently implement parallel applications for high performance computing. Our objective within this work is to propose a generic framework based on adaptive techniques for solving a class of numerical problems on cluster-based heterogeneous hierarchical platforms. Toward this goal, we refer to adaptive approaches to better adapt a given application to a target parallel system. We apply this methodology on a basic numerical problem, namely solving the matrix multiplication problem, while determining an adaptive execution scheme minimizing the overall execution time depending on the problem and architecture parameters.

1 INTRODUCTION

Few years ago, there was a huge development of new parallel and distributed systems. Large collections of interconnected PCs (called clusters) have replaced traditional super-computers in many universities and companies. Due to the increasing performance of on-the-shelf components, such low-cost systems are a reasonable alternative for solving a large range of applications.

However, the introduction of such parallel systems has a major impact on the design of efficient parallel algorithms. Indeed, new characteristics have to be taken into account including scalability and portability. Moreover, such parallel systems are often upgraded with new generation of processors and network technologies. Today, as the systems are composed of collections of heterogeneous machines, it is very difficult for a user to choose an adequate algorithm because the execution supports are continuously evolving. One version will be well-suited for a parallel configuration and not for another. This portability issue becomes crucial because of the frequent changes of the components of the systems. These different elements require to revise the classical parallel algorithms which consider only regular architectures with static configurations and to propose new approaches.

The adaptive approaches are a promising answer to this problem. The idea is to adapt algorithms together with their execution to the target architecture. These algorithms may be automatically adapted to the execution context (data and support).

Our objective within this work is to propose a generic framework including the design of an automatic selection mechanism, based on adaptive techniques for dealing with scalability and portability issues on cluster-based heterogeneous hierarchical platforms for a class of regular numerical algorithms. The proposed methodology may be extended to a class of parallel applications which can be partitioned in a set of independent tasks (which may be non-identical).

The remainder of the paper is organized as follows. We begin in section 2 by presenting the architectural model of the target parallel and distributed system and discussing some adaptive approaches. In section 3, we describe our adaptive framework and detail its components. Section 4 is devoted to a case study where we apply our methodology on the matrix multiplication problem. Section 5 concludes the paper and discusses some perspectives to extend this work.


2 BACKGROUND

2.1 Description of the Architectural Model

We assume in this work a generic model of a platform composed of heterogeneous hierarchical clusters as described in (Capello, 2005). The studied platform enjoys heterogeneity along three orthogonal axes: (a) The processors that populate the clusters may differ in computational powers, even within the same cluster, (b) The clusters composing the platform are organized hierarchically and are interconnected via a hierarchy of networks of possibly differing latencies, bandwidths and speeds. At the level of physical clusters, the interconnection networks are assumed to be heterogeneous, and (c) The clusters at each level of the hierarchy may differ in sizes.

We will extend this architecture to a one where the capacities of the links connecting clusters may change dynamically during the execution of the target parallel application.

2.2 Adaptive Approaches

It is well-known that no single algorithm can always achieve the best performance of a sequential or parallel application for different problem sizes and number of processors on a target parallel system. We can obtain good performances by mixing multiple algorithms for solving the same problem, where each algorithm can dominate the others in specific contexts. Thus, we should determine the more appropriate algorithm (which provides the best performance) in terms of a set of parameters (size of the problem, number of available processors, performances of the interconnection network, etc.), or to combine multiple ones for improving performances to fit well the characteristics of the target computational system. The software mechanism responsible for determining the best available choices at run-time is known as an adaptive algorithm. This mechanism may use different techniques to adaptively determine the best algorithm. For instance, the algorithms presented in (Frigo, 1998; Thomas, 2005) use respectively machine learning and cascading techniques.

3 DESCRIPTION OF THE ADAPTIVE METHODOLOGY

In this section, we describe our methodology for adaptively executing parallel applications in an execution environment characterized by its heterogeneity and its hierarchical organization. An overview of the methodology is sketched in figure 1. The processing is separated in two successive phases. During the first one, we aim to partition the target platform to form subnets of similar characteristics by automatically discovering the network topology. Then, when executing the second phase, we have to determine for each subnet (i.e. cluster) the more appropriate algorithm among multiple algorithmic options leading to the minimum possible execution time of the given problem. We will finally determine an adaptive execution scheme identifying the details of the implementation. It is worthy to note that we may have at a given time different algorithms occurring on different clusters. Moreover, it is possible to use a combination of many algorithms to execute a task on the same cluster. In the sequel, we more detail the major components of the methodology.

![Figure 1: Adaptive methodology.](image-url)

3.1 Partitioning the Platform

Since the target parallel system may be heterogeneous at many levels (computing powers, interconnection network performances, etc.), it is very difficult to manage such platform towards a high performance computing. One way to answer this problem and to minimize the inherent heterogeneity, and thus facilitating the execution, is
to subdivide the network in homogeneous subnets (or logical clusters), as described below. At the end of this phase, we will obtain a set of logical clusters of homogeneous interconnection networks, which will be used to adaptively implement algorithms inside each cluster during the second phase of the methodology.

3.1.1 Network Performance Measurements

The methodology starts by collecting available information from the target execution environment to be used in the step of clustering (see next section). There exist many tools for network monitoring, such as NWS (Network Weather Service). These tools permit to determine many useful parameters of the target parallel system like the current network status, the communication latency, the speeds of the processors, the CPU load, the available memory, etc. For instance, the communication latency and throughput permit to identify groups of machines with similar communication parameters.

3.1.2 Clustering

One reason to construct logical clusters is that machines may behave differently, and the easiest way to optimise communications is to group machines with similar performances (Barchet-Estefanel, 2004). In order to classify nodes in logical clusters, we can use a clustering algorithm similar to the one presented in (Lowekamp, 1996). This algorithm analyses each interconnection on the distance matrix containing the latencies between links in order to group nodes for which their incident edges respect a latency bound (by default 20%) inside that subnet. Note that the distance matrix was obtained when applying NWS on the clusters to determine the network information.

3.2 Adaptive Approach

Once the platform is partitioned in separated homogeneous hierarchical clusters, we have at this stage to determine, using an adaptive approach, the more performant algorithm from a set of algorithms reserved to solve the problem for each cluster. This mechanism may lead to a collection of various methods to be used at the same time on the available clusters. Any necessary characteristics are measured during the first phase corresponding to the network partitioning. We recall that the adaptive decision is made in terms of many information which might be of interest, such as those related to the target problem (size and type of data) and other ones related to the architecture structure (interconnection network, number of nodes, etc.). This phase ends by fixing an execution scheme detailing the implementation. Let us precise that the adaptive algorithm selection is based on analytical models able to predict performances of the parallel application on the target platform.

3.2.1 Strategy of Task Allocation

Assume that we have to execute a set of tasks. Our strategy requires to reserve a node (called coordinator) for controlling the overall execution of the tasks, and one node per cluster to be charged for communicating with the other clusters.

The coordinator starts the execution by assigning a task to each available cluster. Let us recall that clusters may have different performances and tasks may be non identical. Once a cluster finishes the execution of its task, it sends a request to the coordinator to get another task. Then, the coordinator proceeds first by identifying the necessary data to execute the task, which are assumed to be distributed over the platform, their locality (which clusters have the data), and then determining the path minimizing the transfer cost.

3.2.2 The Makespan Improvement Phase

Let us remark that at the end of the execution process, when the number of remaining tasks to execute is less than the number of available clusters, we can observe idle times which may be high especially when the idle clusters are the fastest ones. Figure 2 shows an example where we consider four clusters (C1, C2, C3 and C4). At the instant t1, C4 finishes the execution of its allotted task. Then, it asks the coordinator for another task. We assume at this level that it remains only two tasks to execute. C4 will be allotted a task and the last one will be executed by C3.

At this stage, we propose a strategy inspired from the technique of work-stealing (Blumofe, 1998) in order to reduce the idle time and improve performances. In this case, the first cluster becoming idle at the instant t2 (here is C1) asks the coordinator for identifying which cluster to be concerned with the stealing. The coordinator, having a global status of the execution process, determines the slowest cluster in the sense that finishes last, and decides if it will be performant to share the remaining portion of the task on two clusters (the requester and the slowest, i.e. C1 and C4 respectively). Similarly, the task under execution on C3 will be shared with C2. Let us mention here that sharing the execution is achieved only if this processing is able to reduce the execution time. Figure 2 shows a possible improvement leading to a reduced global execution time.
4 CASE STUDY: MATRIX MULTIPLICATION PROBLEM

We apply our adaptive methodology on a basic numerical problem, namely computing the product of two (large) dense square matrices. We begin by discussing some related works, and then describing our adaptive algorithm.

4.1 Related Works

The parallelization of the matrix multiplication problem was widely studied in the literature. Various optimized versions of this problem have been implemented in libraries on all existing (homogeneous or heterogeneous) parallel systems. We may particularly refer to works presented in (Beaumont, 2001; Desprez, 2004; Hunold, 2004; Lastovetsky, 2004; Ohtaki, 2004) where various methods have been applied, such as standard, fast, mixed, etc. However, only few parallel adaptive implementations have been developed.

However, to the best of our knowledge, no original work has been devoted to implement adaptive algorithms for matrix multiplication on heterogeneous hierarchical clusters where both computing resources and interconnection links are heterogeneous. Moreover, the network capacity may change dynamically during execution. The contribution of this paper is to intend to fill this gap.

4.2 Based Algorithms

We will use three based algorithms in our approach. The first one was designed by Beaumont et al. (Beaumont, 2004) where classical matrix multiplication algorithms have been implemented on heterogeneous clusters. The algorithms presented are very efficient but the distribution used is highly irregular. The second algorithm is proposed by Lastovetsky and Reddy (Lastovetsky, 2004) which have extended a two-dimensional homogeneous block-cyclic distribution to the heterogeneous case that provides perfect load balancing on a grid of processors. The last algorithm is developed by Ohtaki et al. (Ohtaki, 2004) where they propose a recursive data decomposition, which enables both efficient load balancing and incrementing of the recursion level in Strassen's algorithm for heterogeneous clusters. Let us precise that we may use more based algorithms, but we will generate an additional cost for the poly-algorithmic decision.

4.3 Description of the Proposed Adaptive Algorithm

4.3.1 Data Distribution

Let A, B and C=A*B be three square matrices of size n. We assume that due to a previous work, the input matrices A and B are distributed on two different clusters. Our objective here is to distribute the matrices over the available clusters. To compute C, we propose to partition the matrix into equal square blocks of size r each. The size is chosen so that we create coarse-grained tasks that are assigned later to disjoint clusters. Computing a block of C requires a row block of A and a column block of B. So, the initial data distribution is a row block-wise distribution for A and a column block-wise distribution for B. We have now to allocate tasks on (say x) available clusters. The initial allocation that we adopt is presented in figure 3, where we allot cyclically a column of blocks to each cluster. Let us mention that this preliminary allocation will be dynamically adapted during execution.

To be able to start computing blocks, we have to send to each cluster a row block of matrix A and a column block of matrix B. For a new block of the same column block of matrix C, a cluster requires only a row block of A. For a new column block of C allotted to the same cluster, the later requires only a column block of B for the first square block to compute; for the remaining blocks, the cluster has all necessary data. Since the fastest clusters will finish the execution of their allotted tasks first, we will apply the strategy described in section 3.2.2 to

Figure 3: Task allocation on x clusters.
reduce the idle time of these clusters when they have no work to perform.

Remark that at the beginning, all clusters require the same row block of matrix A to start execution. Since the corresponding data is located in a one cluster, we will have to broadcast it on clusters. Each cluster can start its computation as soon as the data is available. We describe in the following how to schedule tasks during the execution.

4.3.2 Scheduling Tasks

Since our problem is reduced here to a set of independent tasks, we will apply the approach proposed in section 3.2. Let us describe the methodology with an example. Consider the platform presented in figure 4, composed initially of three different clusters C1, C2 and C3, with a 1-Port model and two level hierarchy networks with different bandwidths and latencies. After applying the first phase of the methodology corresponding to partitioning the platform in logical homogeneous clusters, C3 will be separated into two sub-clusters C31 and C32. We assume here that the coordinator is a node of C2, matrices A and B are initially located in C31 and C2 respectively, and that the matrix decomposition in blocks leads to 25 tasks to be scheduled on the four clusters.

4.3.3 Adaptive Algorithm Selection

The selection of the best algorithm to execute a given task is based on a performance matrix containing the duration of each algorithm, among the set of input algorithms, on each cluster. This matrix is obtained using analytical models. Formally, assuming a cost model, we denote by $P(Alg_i, C_j)$ the performance of algorithm $Alg_i$ on cluster $C_j$. $Alg_i$ is qualified to be the best on cluster $C_j$ when $P(Alg_i, C_j) = \min\{P(Alg_k, C_j), 1 \leq k \leq q\}$, where $q$ represents the number of available algorithms. It is worthy to note that due to the diversity of clusters composing the platform, we may have at a given time various algorithms executing different tasks, each on a cluster.

Figure 5 shows the scheduling of the different tasks of the example presented previously when the more performant algorithm is used on each cluster. We assumed, since clusters are different in computing powers, that the execution time of a task, which remains unchanged using a given algorithm, is different on two different clusters. We also considered that overhead due to communicating the same amount of data may change from an execution to another due to a possible variation of the network capacity.

5 CONCLUDING REMARKS AND FUTURE WORKS

We have presented in this paper a new (two phase) methodology based on adaptive approaches, including the design of an automatic selection mechanism, for dealing with parallel execution of tasks on heterogeneous clusters.
implementations of a class of regular numerical algorithms and parallel applications which may be partitioned in a set of independent tasks on cluster-based heterogeneous hierarchical platforms. We applied the approach on a basic numerical problem, namely solving the matrix multiplication problem, while achieving the minimum possible execution time depending on the problem and architecture parameters.

As future prospects, we first intend to validate this approach by achieving experiments on real platforms, and apply the methodology on other types of parallel applications. We also plan to integrate other existing adaptive approaches to our framework to benefit from the powerful of these techniques.

REFERENCES


Developing a Fault Ontology Engine for Evaluation of Service-Oriented Architecture Using a Case Study System

Binka Gwynne, Jie Xu
School of Computing, University of Leeds, UK
binka@comp.leeds.ac.uk, jxu@comp.leeds.ac.uk

Keywords: Service-Oriented Architecture, Fault Injection Testing, Ontology.

Abstract: This paper reports on the current progress of research into the development and implementation of a Fault Ontology Engine. The engine was devised to facilitate the testing and evaluation of Service-Oriented Architecture (SOA), using ontologically supported software fault injection testing mechanisms. The aims of this research stem from the importance of system evaluation and the notion that testing and evaluation methods could be better supported for modern distributed systems by autonomous software machines, due to their potential dynamics, size, and complexity of SOA, and the variety of resources they offer. Fault injection testing is still generally in the domain of human expertise, experience and intuition, and machines require knowledge of testing mechanisms to develop testing strategies, with information that is formal, explicit, and in languages that they can interpret. This paper contains descriptions of experimental work carried out in order to generate information for modelling the fault and failure domains of a real-world case study system. Information from this case study system will be used to identify points of interest and target points for subsequent tests. It is hoped to show that inferences taken from known systems can be used for intelligent testing of unknown systems.

1 INTRODUCTION

This research brings together the concepts of ontology, Service-Oriented Architecture (SOA) and fault injection testing in order to develop a Fault Ontology Engine (FOE) with information gained from experiments on real-world case study systems.

Ontologies are formal and logical descriptions of domains used by software to support intelligent communication. There is a great diversity in the design of ontologies and how they represent the real world (Noy et al. 1997), but from a computing perspective, ontology is a logical theory which gives an explicit, partial account of a conceptualisation (Corcho et al. 2003).

An SOA is a type of software architecture comprising services, with emphasis on service interoperability and location transparency, with the aim of achieving loose coupling among interacting software resources. SOA middleware obscures the nature of resources by design, but this can lead to middleware that may be complex and problematic to implement, and systems that are difficult to evaluate.

In consequence novel methods are required for testing and evaluation of SOAs.

Fault injection is a method of testing and evaluating computer hardware and software systems by deliberately inserting them with artificial faults in order to reveal faults more effectively than through observation methods.

2 BACKGROUND

2.1 System Boundaries

A system boundary describes a system’s limit of self-determination and how interactions (inputs and outputs) occur between that system and the external side of its boundary. System boundaries are also concerned with those interactions existing between adjoining systems and how systems interact with their environment: the environment may be considered a system in its own right. System boundaries are important in testing and evaluation because boundary conditions are determinable and
system failures are deemed to be observable deviations from a system’s specified behaviour at its boundary.

2.2 Ontologies

Ontologies are essentially concerned with the nature of existence. They can vary from simple schema to controlled vocabularies, thesauri, hierarchies of terms, taxonomies, to full conceptualisations of domains. Ontologies are commonly used as intelligent communication media for machines (Mizoguchi and Ikeda 1997), and according to Duineveld et al. (1999), ontologies promise: “A shared and common understanding of some domain that can be communicated across people and computers.”

2.3 Service-Oriented Architecture

An SOA is a type of software architecture; it is essentially a collection of services with emphasis on interoperability and location transparency.

According to the World Wide Web Consortium (2006), an SOA is “A set of components which can be invoked, and whose interface descriptions can be published and discovered.”

An SOA is effectively a mechanism for connecting resources on demand by providing a more flexible, loose coupling of components than provided by traditional distributed systems architectures. SOA comprises three basic elements: service provider, service consumer and service repository. A service is defined by how it can be accessed by other components in its system. Access is achieved through a common interface by an exchange of messages with the interface hiding the details of how each service is actually implemented. Services are expected to be self-contained and not expected to have knowledge of the context of other services. Further, the granularity of services is unspecified and can vary, with a service hosted on a single machine or distributed over a network.

SOA middleware is software specifically designed to support interoperable machine-machine interactions over networks and provide common approaches for service definition, publication and use. SOAs can have problems associated with scale and dynamics and its middleware may be complex and difficult to implement (Looker et al. 2005).

2.4 Faults, Errors and Failures

A system failure occurs at a point in time where the condition of a system is different to its expected condition. Errors are responsible for causing failures, and in turn are due to the presence and activity of faults; a fault is a defect of some kind that exists within a system and may be active or dormant. An active fault, or combination of active faults, may produce an error, or combination of errors, whose visibility outside a system boundary is manifested as a failure. This is a well-established, recursive way to describe the relationship between failures, errors and faults (Avizienis et al. 2004). However, although inextricably linked, failures, errors, and faults are conceptually very different entities. A failure is a type of event and a fault a type of cause; both can exist as real-world entities. An error, however, is a type of state and therefore has a virtual existence.

2.5 Evaluating SOA Systems

Organisations offering computing resources need mechanisms to demonstrate the high dependability of their resources in comparison to rival systems. This is especially true for SOA based systems, where the details of service implementations are architecturally hidden.

There are a number of ways to evaluate systems including simulation modelling, stress testing and observation over time. Fault injection is a method of testing and evaluation where artificial faults are deliberately inserted into systems, in order to reveal faults more effectively than through observation (Voas and McGraw 1998), producing errors and observing how they propagate through to failures.

Fault injection is: 1) a faster and more logical way to detect faults than observation; 2) able to test systems at run time; 3) able to test dynamic, distributed systems; 4) allows the use of non-invasive methods such as interception of method calls and data corruption; and 5) able to evaluate how well different types of systems maintain their dependability over time (Arlat et al. 2003, Looker et al. 2005).

3 ONTOLOGICAL SUPPORT

By definition, autonomous software machines are self-managing and can adapt to changing conditions, which means they would provide good support in
evaluation of large or complex distributed systems such as SOAs as these systems can become dynamically modified during tasks. Fault injection is a fast and logical way to detect faults and can be used to test systems at run time, so is a suitable method for testing dynamic, distributed systems. It can also be non-invasive, using mechanisms such as interception of method calls and data corruption. However, fault injection is still generally in the domain of human expertise, experience and intuition, and machines require knowledge to develop their own testing strategies, and require information that is formal, explicit, and in languages they are able to interpret. By using ontologically supported fault injection we aim to: 1) make information on testing and evaluation methods available in machine readable form; 2) improve fault representativeness; 3) avoid spurious testing; 4) lessen the chances of producing undetected spawned faults; and 5) address problems associated with some traditional testing methods such as state and timing.

In our experiments fault and failure ontologies are used to guide more targeted fault injection tests through the analysis of cause and effect pairs as experimental data, including provenance, provides the information for FOE. In addition, is intended that FOE will be used in future in combination with other tried and tested software fault injection tools designed for SOA, such as WS-FIT (Looker et al. 2005).

4 A CASE STUDY

4.1 Secure Power System

A manufacturer of battery based AC and DC secure power systems was chosen for the case study. The principal markets for their products are industries involved in telecommunications, power generation and distribution, and petrochemicals. Uninterrupted, secure power units are vitally important to these industries, for example, the failure of PA systems on oil rigs was implicated as partially responsible for the loss of life in the Piper Alpha disaster, 1988 (UK Health and Safety Executive 2002).

The case study system is a secure power system that maintains a power supply for mobile telecommunications. The control units are mainly located on roof tops, are subject to sporadic mains power failures for generally short periods of time, with unit housing often subject to high ambient temperatures. The control units are remotely monitored.

These secure power systems can be maintained by one of two types of battery: Valve Regulated Lead Acid (VRLA) and Nickel-Cadmium (Ni-Cad). The VRLA battery is the newer technology, comprising a sealed system which does not require topping up by hand. However this battery requires more monitoring than the Ni-Cad battery due to physical vulnerabilities, although monitoring can be carried out remotely. In order to take advantage of the low maintenance required for VRLA, it is necessary for the system to maintain a high level of reliability through reliable monitoring.

Typical information monitored is temperature, voltage levels in batteries / cells, power input and power output to load etc., but monitors can also warn of such things as a door to the unit housing being open (a potential source of damaging heat input).

4.2 Fault Model

System fault and failure taxonomies were heuristically drawn up from known information on how the case study system behaves. Elements in the fault model were predicted failure-value pairs. For example: a voltage “Stuck-at” fault; this could lead to the alarm not sounding when the condition point is reached and the switch not closing to load at EOD and the hardware being damaged. The outcome of this example may be that the fault is sometime later observed; the failure could be classified as “Observed failure, non-recoverable, known cause, known effect”, and the scenario fault-failure pair would be “voltmeter stuck-at fault” and “hardware, cells/battery damaged”.

4.3 Experiments

A faulty voltmeter was simulated in the secure power system. The simulation involved injecting faulty values into a data set with the results logged in XML-based files. Each test carried out was given a unique ID and information was stored in logs, in order that information would be available and tests replicable later.

When fault injection testing is carried out there is an intention to guarantee that failures occur so that they can be observed. However, naturally occurring or programmed fault tolerance in a system can sometimes make it difficult for failures to occur where only one fault is present and it may become necessary to use a combination of faults, or composite fault, to bring about an observable failure. Consequently, experiments were also carried out that
simulate this situation, with each composite fault paired with its respective outcome.

![Figure 1: Injected Fault - Voltmeter Data.](image)

In the next experiment the sensor tested was the thermometer for the unit housing, which is monitored in order to prevent damage to battery cells from high temperatures. An alarm fires when the temperature approaches 20 deg. Celsius, caused by a hardware fault in the unit housing, which leads to a temperature increase in the system. In this scenario the assumptions are that the sensor (thermometer), alarm and housing are all functioning correctly. The fault is in the refrigeration unit and for a short period of time the temperature exceeded 20 degrees C, the consequence is that there is slight hardware damage sustained before the fault is rectified externally (by human intervention). The fault is recovered, due to the correct function of the alarm and monitor, and some external action occurring to rectify the problem when the alarm sounds.

5 FUTURE STUDY

Information from these experiments is being analysed and stored in machine readable form, so that FOE can use it to predict future points of interest in the system fault domain. These points of interest will form the target points for further tests as FOE uses its information to make decisions on the next phase of testing and evaluation.

The information obtained from injecting faults with predictive outcomes into this known system is leading on to the design of further test scenarios to discover more of this system’s fault and failure domains. It is hoped to build on to this information to develop new testing strategies for unknown systems and a new mechanism to glean information about their fault and failure domains.

Further experiments will be carried out that continue with the case study, and a new case study will be introduced, to determine where inferences from the first case study are useful to discovering information about other systems. This should demonstrate that experiments on known systems can be used to investigate the fault and failure domains of unknown systems. The next progression in this research is to use this information to test and evaluate the reliability of SOA-based systems.

REFERENCES


AN APPROACH TO MULTI-AGENT VISUAL COMPOSITION WITH MIXED STYLES

Joaquim Reis
Departamento de Ciências e Tecnologias de Informação, ISCTE, Avenida das Forças Armadas, 1600 Lisboa, Portugal
Joaquim.Reis@iscte.pt

Keywords: Agents, Creativity, Design, Shape Grammars.

Abstract: Applications of computer systems that mix Art, Science and Engineering have appeared as a result of the evolution of information technologies in the last three decades. Frequently, they involve the use of Artificial Intelligence techniques and they have appeared in the fields of music, literary arts and, more recently, visual arts. This article proposes a computational system based on creative intelligent agents that, by making use of the shape grammar formalism, can support visual composition synthesis activities. In this system, each agent gives its creative contribution through a style of its own. Different modes of agent contribution can be put into perspective like, for instance, cooperative or non-cooperative modes, the resulting composition emerging from these contributions.

1 INTRODUCTION

We can say that the degree of evolution of the human species can be measured by the degree of sophistication of the technology it uses, as well as by the degree of social organisation. There has always been a relationship among technology evolution, human evolution and human societies evolution. The importance of information technologies has been increasing progressively. Presently they impact in every area of society, not only in engineering or management, just to cite the most evident cases, but, and more recently, in art too.

Of course that any form of art involves always some kind of technology. To produce his work, the painter uses tools like brushes, dyes, canvas, for instance, as well as certain techniques to work with these tools. But, in this era of information and computers, the mix of technology and art is becoming more and more common, not just at the level of execution, but also at the level the creative process. The contribution of Artificial Intelligence, an area of computer science born in the 1950s (Rich 1991, Russell 2003), has been significant for this.

This article makes a very brief revision of the application of computer systems to the artistic creative field, next centers around the visual arts, then introduces the theme of shape grammars, and finally it proposes a system based on creative agents to support the visual composition activity.

2 ART, COMPUTERS AND ARTIFICIAL INTELLIGENCE

There has been, in the last three decades, an increasing use of information technologies in the musical arts. Due to the digital technology of computers it is nowadays possible to store, analyse, modify and synthesize with an accuracy higher than that of the human ear system, and even generate musical compositions. It was in the musical arts that information technologies have penetrated more rapidly, perhaps due to the facts that music is of a more quantitative nature, was more theoretically mature and the needs of memory and processing capacity aren’t excessive (at least when compared to those of visual arts). With the progression of Artificial Intelligence and the appearance of the knowledge based systems, in particular expert systems, new methods of musical composition have appeared. The process of composition now involves an interaction between the composer and the computer system in which the composer generates, and concentrates on, the original ideas for a composition, and has support from the computer system that was programmed with knowledge about the composition process (Kurzweil 1990, Miranda 2001).

In the literary arts computers are presently very useful. For instance, text processors are indispensable software nowadays and together with them, there are also a set of software tools like
spelling correctors, grammar correctors, style correctors, and dictionaries, thesaurus and other kinds of linguistic databases. In particular, in the field of natural language processing, Artificial Intelligence has had an important role (Winograd 1983, Allen 1987), in understanding (analysis) as well as in generation (synthesis). Automatic translation, data base interfaces in natural language and other kinds of systems involving dialogue, story understanding and generation and poetry generation, are some examples of the rich contribution of Artificial Intelligence.

In the visual arts, due to a slower evolution of hardware, in particular respecting to graphic capabilities (limited graphic resolution of output devices), only more recently (when compared to the musical and literary arts) the use of information technologies has become more attractive. From the decade of 1990 on we could watch a progression in the investigation, with its products migrating progressively to software tools to support the creative activity of designers, architects and graphical artists in general. The computer supported visual arts methods of creation vary from free hand drawing by using the computer screen as it were paper or canvas, to the most complex image processing involving the generation of shadows, surfaces, shapes, colors, and the execution of translation, rotation, scaling repetition and distortion operations on shapes. Mixing with the cognitive aspects, in the field of the creative process, mathematical processes and Artificial Intelligence techniques like fractals, chaos theory, genetic algorithms, rule based systems and artificial life (Kurzweil 1990), the latter related to the idea of agents of Distributed Artificial Intelligence (Ferber 1999), have been used. Some of these techniques, namely artificial life, have migrated to the animation and cinema field. As examples of the application of these methods to the visual arts we can point the works of the painter Harold Cohen, with his AARON program (Cohen 1999), and of Leonel Moura with his system of painter robots, that were both participants in an recent art exposition in Lisbon (Bioart 2005).

3 THE GRAMMARS OF SHAPE

Shape grammars were introduced in the 1970s by Stiny and Gips (Stiny 1972). They are similar in principles to the grammars used in the area of Artificial Intelligence in natural language understanding and generation (Allen 1987), with the difference of being based not on symbols, but on shapes (points, lines, two-dimensional and three-dimensional geometric shapes) as well as, by extension, also other parameters like dimensions, colors, etc.

A shape grammar is composed by a basic vocabulary of shapes, an initial shape and a finite set of rules that specify how shapes can be generated from other preexistent shapes, similarly to the lexicon, the initial symbol and the grammar rules of a language used in a natural language processing system. The rules of a shape grammar specify how, in a composition in progress, shape existing in the composition can be replaced by new shapes.

Each rule has a left side, pre-condition, or antecedent, and a right side, action, or consequent. The left side specifies the pattern for which the rule is applicable and the right side the respective pattern for substitution. Briefly, a rule is applicable if there is a similarity transformation (i.e., an isometry or a transformation of scale) leading to a match of the shape of the left side of the rule with a shape existing in the composition. Rules are applied in a forward manner (from antecedent to consequent), like in the production/rule-based forward-chaining systems of Artificial Intelligence, which perform a kind of forward inference. When applied, a rule replaces the shape(s) matched in the composition with the shapes in its right side. Symbolically, we can express a rule in the following schematic form:

\[
\text{<shape(s)-to-match>} \Rightarrow \text{<shape(s)-for-substitution>}
\]

Special markers (labels) that aren’t part of the composition can be used for rule application control and application termination condition specification.

\[
\begin{align*}
\square & \rightarrow \text{ } \square \\
\square & \rightarrow \text{ } \square \\
\square & \rightarrow \text{ } \square \\
\square & \rightarrow \text{ } \square \\
\end{align*}
\]

\[\text{...}\]

(a) rule \hspace{1cm} (b) resulting composition

Figure 1: Shape grammar example.

For instance, in Figure 1 we show a shape grammar with only one rule, as well an example of a possible result from the repeated rule application.

Even a simple shape grammar like this can show some emergence behavior. This feature depends on the degree of detail and hierarchy accommodated by the internal computational representation used for the geometric shapes in the system (which influences the ability to consider and process certain shapes as non-atomic ones, and to recognise shapes in the composition that weren’t explicitly included in it).
The use of shape grammars has been exploited in applications in different design problems, in the context of sythesis (generation) and analysis (interpretation) of visual compositions and also as means to the description and the representation of styles, including for didactic purposes and also other specific applications, for instance in architectural drawings (Gips 1999, Tapia 1999, Knight 2000, Mitchell 1998).

A style is a way of someone doing something (Simon 1971) and shows up when that someone chooses an alternative or a process for generating a solution. In the field of design, a style is a king of design knowledge which is a characteristic of a product, or a set of products, of design, and is recognisable through the presence of some visual elements like shape, color, relative position, texture, dimension, orientation (Dondis 1973, Bonsiepe 1983, Wong 1993), as well as certain ways of combining those elements. Visual compositions can be generated automatically according to specific styles. Each style is implemented by the set of rules of the shape grammar specific to the style.

4 AGENTS, STYLES AND AGENTS WITH STYLE

The Intelligent Agents technology has been applied to complex problems involving intelligence and interaction among agents, human included, namely in the field of the Internet and in animation in cinema. It is an intelligent systems technology from the area of Artificial Intelligence that proposes a problem solving approach where the problem solving effort is distributed to a group of intelligent agents (Weiss 1999, Ferber 1999). In this context, an agent is defined as an entity situated in an environment, that perceives the environment, and acts in the environment. Not rarely, an agent based system is composed by a certain number of agents, i.e., it is a multi-agent system, each with a certain level of intelligence and a certain level of communication capabilities. In terms of problem solving, the idea behind a multi-agent system is to approach the solution of problems that an isolated agent wouldn’t be able to solve.

Respecting to the complexity of the internal architecture, we can classify agents in cognitive and reactive agents. Cognitive agents typically have an architecture containing internal representation mechanism that allow them to maintain an up-to-date model of their environment and of themselves, to reason about the environment and
about the results of its own actions in the environment. This kind of agents is the most sophisticated, autonomous and intelligent, and they have capabilities to maintain goals and plan their actions in order to be able to attain those goals. In general, multi-agent systems with cognitive agents are composed by a small number of agents and the activity in the system comes more from the cognitive activity of the agents than from the interaction amongst them. Communication can be very sophisticated and is usually based on message passing, and may include standardised protocols. Agent activity coordination mechanisms can vary and can go from a total cooperation among agents to antagonistic forms involving competition and negotiation.

Reactive agents have a very simple architecture. They don’t have any internal representation and reasoning mechanisms (and thus, they can’t have goals, and they aren’t able to plan their actions) and its behavior is based on stimulus-response patterns. In general, multi-agent systems based on reactive agents are composed by a great number of agents and the activity of the system stems more from the interaction amongst them. Communication is very primitive, and is carried through signal exchange or by operating and recognising changes in the environment.

The system we propose in this article is a multi-agent system based on creative Intelligent Agents in which shape grammars are used to support an activity of visual composition. In this system, sketched in Figure 4, each agent (see Figure 5) gives its own creative contribution through a style of its own. A style is represented by a set of rules implementing the shape grammar for the style. Each agent has the set of rules of its own style, and tries to apply them, depending of the present state of the composition. Given a set of these agents, each one with its own style, and given a set of shapes recognisable in the context of those styles each agent will apply the rules of its style, whenever possible, together progressively generating a visual composition, or a set of alternative visual compositions.

As the agents of this system are rule-based they are, essentially, reactive. However, the rules contain knowledge, i.e., problem domain knowledge (styles of visual composition) so, in a certain perspective, the agents can be considered cognitive. The interaction among the agents occurs through changes in their environment, which means the visual composition in progress. Control of the agent activity can involve human external intervention. Several forms of agent activity coordination can be put in perspective, some involving more cooperative agent contributions others more competitive in nature. The resulting composition will emerge from these contributions.
other of a more cooperative one (in this case the result of the style application shows more order).

Possible applications for this kind of system are visual composition generation with mixed styles, free generation or controlled and goal oriented generation (e.g., technical drawing, Web page layout design).

5 WORK IN PROGRESS AND FUTURE WORK

The system we have described is still being conceptualised but there is already some work in progress leading to its realisation. This work in progress includes an embryo of an ontology of a geometric two-dimensional domain, including geometric points and relative position relations between geometric points on the x and on the y direction, line segments and relative position relations between line segments on the x and on the y direction, and rectangular shapes and relative position relations between rectangular shapes on the x and on the y directions in the plane. In the following we very briefly show some aspects of this ontology.

![Relative position relations between two geometric points on the same line.](image)

Figure 7: Relative position relations between two geometric points on the same line.

Starting from the relative position relations between two geometric points on the same line shown in Figure 7, we have built the 13 basic relations between two line segments horizontally (x axis) and vertically (y axis). For reasons of space economy we only show, in Figure 8, the relations for the first case (x axis); for the other case there is an equal number of similar relations but in the vertical direction (y axis). These relative position relations on an axis are inspired by the temporal interval relations of (Allen 1983).

Combining these relations pairwise we can get the relative position relations between rectangular shapes in the two-dimensional plane, in number 169 (13x13).

![Relative position relations between two line segments horizontally in the x axis.](image)

Figure 8: Relative position relations between two line segments horizontally (in the x axis).

For reasons of space economy we only show, in Figure 9, 13 of those relations.

![Relative position relations between two rectangular shapes in the two-dimensional plane.](image)

Figure 9: Relative position relations between two rectangular shapes in the two-dimensional plane.

A different aspect in the work in progress is a small prototype program, being developed in
Common Lisp, to implement the forward-chaining inference engine for experimenting with the set of rules the shape grammar of an agent. We also intend to look for software tools that allow suitable domain knowledge representation and reasoning, preferably implemented in, and with good integration with the symbolic environment of, Common Lisp. Possible candidates are the rule and logic based tools LOOM (LOOM 2005) and LISA (LISA 2005), for instance.

In the near future, more immediate work to be developed has to do with computational geometry (e.g., geometric shape manipulation, including similarity transformations, particularly for recognizing shape patterns in the left side of shape grammar rules). But the most significant aspects to give special attention to are the control mechanisms for rule application, which can be viewed in the intra-agent and in the inter-agent perspectives, the latter involving the coordination of the activity of the agents in the multi-agent system.

REFERENCES


Knight 2000. Knight, Terry, Shape Grammars in Education and Practice: History and Prospects, Department of Architecture, MIT, 2000 (http://www.mit.edu/~tknight/IJDC/).


A PEER-TO-PEER SEARCH IN DATA GRIDS BASED ON ANT COLONY OPTIMIZATION

Uroš Jovanovič
xlab Research
Teslova 30, SI-1000 Ljubljana, Slovenia
Email: uros.jovanovic@xlab.si

Boštjan Slivnik
University of Ljubljana, Faculty of Computer and Information Science
Tržaška 25, SI-1000 Ljubljana, Slovenia
Email: bostjan.slivnik@fri.uni-lj.si

Keywords: Distributed search, P2P, data grids, ant colony optimization.

Abstract: A method for (1) an efficient discovery of data in large distributed raw datasets and (2) collection of thus procured data is considered. It is a pure peer-to-peer method without any centralized control and is therefore primarily intended for a large-scale, dynamic (data)grid environments. It provides a simple but highly efficient mechanism for keeping the load it causes under control and proves especially useful if data discovery and collection is to be performed simultaneously with dataset generation. The method supports a user-specified extraction of structured metadata from raw datasets, and automatically performs aggregation of extracted metadata. It is based on the principle of ant colony optimization (ACO). The paper is focused on effective data aggregation and includes the detailed description of the modifications of the basic ACO algorithm that are needed for effective aggregation of the extracted data. Using a simulator, the method was vigorously tested on the wide set of different network topologies for different rates of data extraction and aggregation. Results of the most significant tests are included.

1 INTRODUCTION

Nowadays, vast datasets too large to be stored on a single computer are being generated and used all the time. In the scientific environment, they are often produced during experiments in a number of different fields, ranging from physics to genetics. In business, companies are storing all kinds of data from which customer behaviour can be analysed and predicted. In computing and telecommunications, logs are generated day and night.

Some datasets are generated by a single source but must be distributed immediately because of their size. For example, data grids are being developed to manage all data produced by particle accelerators. Other datasets like computer logs, for example, are generated by multiple sources but are most often analysed together.

Data grids represent a promising way to handle large and distributed datasets (Chervenak et al., 1999; Berman et al., 2003). To enhance the overall system performance, data replication is used to minimize the transfer of different chunks of a distributed dataset to the computer running the application (Čibej et al., 2005).

However, if the amount of data needed by the application is small compared to the entire dataset, the data should be extracted at the remote servers, collected together, and finally transferred back to the application. Thus, an application must be capable of sending custom requests to the dataset servers. In a distributed environment based on an unstructured network with no fixed topology and no metadata prepared in advance, this problem is hard to solve efficiently.

Grid infrastructure like Globus or gLite (Globus, 2006; gLite, 2006) is used to handle all low level details of sending the requests and obtaining the results, and to provide read access to raw data in order to extract metadata. To avoid flooding the network with requests and thus keeping the load limited the implemented grid service uses a method of ant colony optimization for extracting and collecting data from chunks of a distributed dataset (Dorigo and Stützle, 2004).

It has been demonstrated that ACO can be an effective method for clustering (Handl et al., 2006), but so far only static datasets have been considered. In this paper the ACO algorithm, adapted and extended in order to handle extraction and collection from dynamic raw datasets, is described.
2 USER’S PERSPECTIVE

Following the approach introduced in (Jovanović et al., 2006), a grid service must enable a user to do the following two tasks:

1. **Starting a new search:** As a user is given full (read) access to the raw dataset, a user should be able to specify a program that searches the local chunk of a distributed dataset and extract the relevant data.

2. **Checking the list of performed searches:** A user must be able to check if any other user has already performed a search equal or similar to the one he or she is about to start.

As the user’s program extracts data from the local chunk only, it is the responsibility of the grid service to transfer the user’s program to each dataset server, to run the program, and to collect the extracted data from all servers.

The first reason for keeping a list of performed searches is to keep the overall system load low (a) by not repeating the same search over again and (b) by using results from a similar (possibly more general) search instead of starting a new one. The second reason is a fact that the list of performed searches acts as a virtual blackboard about what data or relations among data in the dataset seem interesting to other users. This might be especially valuable in a collaborative scientific research. It follows that the results of the searches performed in the past should be stored in one form or the other somewhere in a data grid.

Hence, the typical method for performing a search is best described by an (informal) Algorithm 1. Note (1) that starting a new search in line 6 does not block the execution as the new search is performed on remote servers; (2) that the search and its results will eventually appear on the list of performed searches (even if it yields no results) and thus the loop in lines 7–13 does terminate.

**Algorithm 1 Performing a search.**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>check the list of performed searches</td>
</tr>
<tr>
<td>2</td>
<td>if the (similar) search has been found then</td>
</tr>
<tr>
<td>3</td>
<td>return its results and stop</td>
</tr>
<tr>
<td>4</td>
<td>end if</td>
</tr>
<tr>
<td>5</td>
<td>provide the extraction program</td>
</tr>
<tr>
<td>6</td>
<td>start a new search</td>
</tr>
<tr>
<td>7</td>
<td>loop</td>
</tr>
<tr>
<td>8</td>
<td>check the list of performed searches</td>
</tr>
<tr>
<td>9</td>
<td>if the search has been found then</td>
</tr>
<tr>
<td>10</td>
<td>return its results and stop</td>
</tr>
<tr>
<td>11</td>
<td>end if</td>
</tr>
<tr>
<td>12</td>
<td>sleep for a specified amount of time</td>
</tr>
<tr>
<td>13</td>
<td>end loop</td>
</tr>
</tbody>
</table>

This method allows different implementations of a distributed index of performed searches and different implementations of searching. At the time being, the research is focused on the efficient peer-to-peer implementation of a single search (line 6). The task of maintaining a list of performed searches in a distributed index and especially a method for comparing descriptions of different searches are left for the future work.

3 ANT COLONY OPTIMIZATION FOR SEARCHING RAW DATASETS

Ant colony optimization is a biologically-inspired optimization method (Dorigo and Stützle, 2004). The basic idea is to use a large number of simple agents called ants: each ant performs a relatively simple task but combined together they are able to produce sophisticated and effective optimization. Further improvements of ACO are based usually on a combination of the ACO algorithm with other local optimization techniques (Dorigo and Stützle, 2004). Ant based clustering and sorting have already been studied in the past (Deneubourg et al., 1991; Handl et al., 2006).

There are two main differences between ant based clustering and peer-to-peer searching as described in this paper. First, a node in the system can possess a pile of data, not just one datum. Second, other ant based clustering algorithms are suited for mesh or similar regular planar topologies, and thus the distance function is modified in order to be efficient for an arbitrary topology of a distributed environment.

**Extraction Ants.** Using user-specified programs, extraction ants extract data from local chunks of raw datasets. Each extraction program is uniquely identified. Extraction ants mark their paths with pheromones associated with the id of the program they are carrying. In order for extraction ants to discover as much data as possible they avoid the trails and prefer the clean paths.

**Aggregation Ants.** The data extracted by extraction ants is collected into piles by aggregation ants. Aggregation ant is based on Algorithm 2. According to the pick-up function \(1/(1 + x/n)^k\), small piles of data are picked up with a very high probability while large piles are most unlikely to be picked up. The distinction between small and large pile is predefined and based on the characteristics of the environment, such as the average connection bandwidth, and expected amount of data. Note that the distinction between a small and large piles can be simply regulated by changing \(n\) (a measure for the size of a pile) and \(k\)
ant has a load
dropped load
the gap on the trail
prevents formation
of the cycle
after the gap,
the trail continues

Figure 1: Result of the acyclic path marking. (The trail on the right is a “backward” trail produced when the ants is moving away from the location where a data has been dropped.).

(a measure of strictness) in the pick-up function. Second, the dropping probability function is simplified: the ant decides to drop the load whenever data of the same type is present on the node. Furthermore, in order to limit the network load, the number of hops that a loaded ant can take is limited.

The probability of finding some data is based on the number of pheromone trails that lead to the pile containing the data. To increase this probability, after dropping the load, an ant marks the first few connections with pheromones directed to the location of the dropped data. The outcome of such pheromone marking is a tree of pheromone trails where every branch of the tree is directed towards the root of the tree that contains a pile of data. The result of such process is shown in Figure 1.

There are positive as well as negative consequences of these so called pheromone fields. On the positive side, there is a high probability that a smaller pile is correctly merged with larger pile. On the negative side, once an ant enters the pheromone field, it gets trapped. The ant that picks up a pile gets trapped in the pile’s own pheromone trails.

Pile Ants. In order to avoid the pheromone fields because of their negative effects, pile ants are used. They are simple random walkers that are produced by piles and ignore any pheromone information. At every new location, a pile ant checks if another pile of data of the same type as the one in the pile that emitted it, exists. The smaller pile is picked up and merged with the bigger pile.

One-Time Aggregating Ants. One-time aggregation ants are special type of aggregation ants. Whenever an extracting ant extract some data, a one-time aggregation ant is also created at the same location. This ant picks up the new extracted data and tries to drop it at an appropriate place. When the data is dropped, the ant dies, i.e., is removed from the system. One-time aggregation ants annul the time needed to discover new extracted data and increase the probability that they are dropped into near piles.

Algorithm 2 A pheromone-based aggregation ant.

1: loop
2: while the ant is empty do
3: while the node is empty do
4: select a random direction
5: make a step in the selected direction
6: end while
7: if load is selected then
8: pick up the load; $h \leftarrow 0$
9: end if
10: end while
11: repeat
12: select a direction using the existing pheromones
13: mark the selected direction
14: make a step in the selected direction
15: $h \leftarrow h + 1$
16: if $h = h_{\text{max}}$ or the node load = the ant load then
17: drop the load; $h \leftarrow 0$
18: end if
19: until the ant is empty
20: while $h < h_{\text{max}}$ do
21: select a random direction
22: mark the selected direction using the cycle-prevention method
23: make a step in the selected direction
24: $h \leftarrow h + 1$
25: end while
26: end loop

Query Ants. The role of the query ants is to find data collected in piles by following the pheromone paths created by aggregation ants.

Another method for data discovery excludes the need for query ants. It is based on maintaining of distributed indexing service. When the pile is static and big enough, it registers its location into this distributed indexing service. Here, static property of a pile means that the pile has not changed its position in some predefined amount of time. The second property, being big enough, means that the pile contains at minimum some predefined amount of aggregated metadata.
4 EXPERIMENTAL RESULTS

The experimental results were obtained by simulating the extraction and collection of data using a two-layered network consisting of 2550 nodes. There are 50 fully-connected nodes, each of them being a node of a subnetwork consisting of another set of 51 fully-connected nodes. We have also performed the tests on different layouts and of different scale, but different topologies yield very similar results to those presented here.

For the chosen topology and the number of nodes, we tested our algorithm against the random walk heuristic. Note that during the extraction the aggregation is also being performed. We have also tested the scenario when the extraction stops and only aggregation is being performed. These cycles are referred to as pure aggregation cycles.

Figure 2 shows the results obtained when data is being extracted on 100 random locations in each iteration, and 300 aggregation ants were always present in the system besides one-time aggregation ants.

5 CONCLUSION

In this paper we have presented modifications of basic ACO algorithms that are well suited to the limitations of the distributed environments. The described modification of ACO enables very simple yet precise runtime control over the load caused by extraction and aggregation simply by regulating the number of ants.

REFERENCES


AUTHOR INDEX

Anderson, C. ............................................................ 13
Apel, S. .................................................................. 127
Athanasopoulos, G. ............................................. 203
Bae, D. ................................................................. 257
Bamha, M. ........................................................... 301
Bergel, A. ............................................................. 29
Biagio, C. ........................................................... 218, 253
Bouguerra, S. ..................................................... 345
Boulanger, F. ...................................................... 247
Brown, R. ............................................................ 151
Calero, C. ............................................................ 238, 333
Cantone, G. ......................................................... 107, 218, 253
Cardona, M. ....................................................... 133
Cazzola, E. ........................................................... 5
Cazzola, W. ......................................................... 263
Clarke, S. .............................................................. 29
Coleman, G. ........................................................ 81
Constantines, C. ................................................ 177
Correa, M. .......................................................... 13
Costanza, P. .......................................................... 29
Crespo, Y. ............................................................ 165
Damiani, F. ........................................................... 5
Daubner, B. .......................................................... 232
Escalaona, M. ........................................................ 283
Falcarin, P. .......................................................... 171
Falessi, D. ............................................................ 107
Ferreira, D. ........................................................... 145
Fournier, A. .......................................................... 177
Garcia, F. .............................................................. 224
Ghoneim, A. ........................................................ 263
Giachino, E. ........................................................... 5
Giambiagi, P. ........................................................ 339
Gianni, A. .............................................................. 5
Gómez, O. ............................................................. 224
Gordea, S. ............................................................ 210
Greaves, D. .......................................................... 71
Guirado-Puerta, A. ............................................. 319
Gutiérrez, J. ........................................................... 283
Gwynne, B. ........................................................... 353
Hassan, M. ........................................................... 301
Henrich, A. ........................................................... 232
Henriques, D. ........................................................ 46
Hirschfeld, R. ........................................................ 29
Hong, J. .................................................................. 257
Hyland, P. ............................................................. 273
Ichimori, T. ............................................................. 39
Jeon, S. .................................................................. 257
Jovanović, U. .......................................................... 363
Katifori, A. ........................................................... 273
Kawada, S. ............................................................ 190
Kelsen, P. .............................................................. 63
Kristensen, B. ........................................................ 54
Kuang, H. .............................................................. 196
Kuchen, H. ............................................................ 291
Kuhlemann, M. .................................................. 127
Kusakabe, S. ......................................................... 308
Law, W. ............................................................... 159
Lee, H. ................................................................. 257
Leich, T. ............................................................... 127
Lepours, G. ............................................................ 273
Lomartire, A. ......................................................... 253
López, C. .............................................................. 165
Manzanares-López, P. ........................................... 319
Marticorena, R. ................................................... 165
Martinez-Marchena, I. ......................................... 133
Masuyama, H. ........................................................ 39
McCaffery, F. ........................................................ 81
Mejías, M. ............................................................ 283
Moraga, M. ........................................................... 333
Mora-López, I. ........................................................ 133
Muñoz-Gea, J. ....................................................... 319
Nasri, W. .............................................................. 345
Newmarch, J. ........................................................ 89
O’Connor, R. ........................................................... 81
Ohnishi, A. ............................................................ 279
Ohta, T. ................................................................. 190
Oktaba, H. .............................................................. 224
Ormandjieva, O. .................................................... 196
Ove, O. ................................................................. 339
Pantazoglou, M. .................................................... 203
Paquet, J. .............................................................. 196
Parsamanesh, P. .................................................... 177
Pennella, G. .......................................................... 218, 253
Pérez, F. ............................................................... 165
Perez, J. ............................................................... 118
Pesce, G. ............................................................... 218
Pfeifer, M. ............................................................. 98
Piattoni, M. .......................................................... 118, 224, 238, 333
Piper, I. ................................................................. 151
Poldner, M. ........................................................... 291
Puntigam, F. ........................................................... 21
Ravn, A. ............................................................... 339
Reis, J. ................................................................. 325, 357
Ruiz, F. ................................................................. 118
Ruiz, J. ................................................................. 238
Saake, G. .............................................................. 263
Sanchez-Aarnoutse, J. ........................................... 319
Sanchez-Aarnoutse, J. ........................................... 319
Sasama, T............................................................... 39
Sawyer, P.............................................................. 139
Schmid, H............................................................ 98
Schneider, G...................................................... 339
Sequeira, M......................................................... 46
Serrão, C............................................................. 46
Shimokura, M..................................................... 190
Silva, A............................................................... 145
Slivnik, B............................................................ 363
Song, I............................................................... 257
Stone, A.............................................................. 139
Taniguchi, H....................................................... 308
Tarawneh, H....................................................... 269
Torchiano, M....................................................... 171
Torres, J............................................................. 283
Tsalgatidoum, A.................................................. 203
Tsen, C............................................................... 13
Vassev, E........................................................... 196
Vassilakis, C....................................................... 273
Vidal-Naquet, G.................................................. 247
Videira, C........................................................... 145
Walker, D........................................................... 333
Westfechtel, B.................................................... 232
Xu, J................................................................. 353
Yamada, S.......................................................... 308
Zanker, M.......................................................... 210
Zenida, P............................................................ 46