Conferences in Research and Practice in Information Technology

Volume 48

Computer Science 2006

Australian Computer Science Communications, Volume 28, Number 1.

AUSTRALIAN COMPUTER SOCIETY

acsw

COmputing Research & Education

Computer Science 2006

Proceedings of the
29th Computer Science Conference (ACSC 2006),
Hobart, Australia, 16-19 January 2006

Vladimir Estivill-Castro and Gillian Dobbie, Eds.

Volume 48 in the Conferences in Research and Practice in Information Technology Series.
Published by the Australian Computer Society Inc.

Published in association with the ACM Digital Library.

Proceedings of the Twenty-Ninth Australasian Computer Science Conference (ACSC 2006), Hobart, Australia, 16-19 January 2006

Conferences in Research and Practice in Information Technology, Volume 48.

Editors: Vladimir Estivill-Castro
School of Computing and Information Technology
Nathan Campus
Griffith University
Brisbane 4111
QLD Australia
Email: `v.estivill-castro@griffith.edu.au, estivill-castrov@acm.org`

Gillian Dobbie
Department of Computer Science
University of Auckland
Auckland,
New Zealand
Email: `gill@cs.auckland.ac.nz`

Series Editor: John F. Roddick,
Conferences in Research and Practice in Information Technology
Flinders University,
PO Box 2100, Adelaide 5001
South Australia.
`crpit@infoeng.flinders.edu.au`

Publisher: Australian Computer Society Inc.
PO Box Q534, QVB Post Office
Sydney 1230
New South Wales
Australia.

Conferences in Research and Practice in Information Technology, Volume 48.
ISSN 1445-1336.
ISBN 1-920-68230-9.

Printed, November 2005 by Flinders Press, PO Box 2100, Bedford Park, SA 5042, South Australia.
Cover Design by Modern Planet Design, (08) 8340 1361.

The *Conferences in Research and Practice in Information Technology* series aims to disseminate the results of peer-reviewed research in all areas of Information Technology. Further details can be found at http://crpit.com/.

Table of Contents

Preface

Edsger Dijkstra, was a pioneer of the field of Computer Science and a participant in 1977 in the first Australasian Computer Science Conference later jovially named as ACSC-0. He is credited for indicating that *Computer science is no more about computers than astronomy is about telescopes*. We understand now that computer science is the accumulated knowledge through scientific methodology of data and information manipulation by the use of the computer.

The 29th Australasian Computer Science Conference (ACSC-2006) was held at the School of Computing at the University of Tasmania, Hobart, Tasmania, Australia from January 16th to 19th, 2006. It is part of the Australasian Computer Science Week and brings several parallel conferences together. ACSC-06 represents a strong and reputable academic meeting addressing many research sub-disciplines in Computer Science. It brings together the international community with a central location around Australia and New Zealand. The meeting allows academics and researchers to discuss research topics as well as the progress of the field and policies to stimulate its growth. It encourages the dissemination of ideas in an inter-disciplinary and intra-disciplinary fashion. Researchers that have specialized in one direction find and audience in others that have moved in orthogonal directions. Thus, the conference offers a forum for many topics (over 40 general topics were listed in the Call for Papers). The International Program Committee integrated expertise in all these areas as well as representation from most of the Australian and New Zealand Higher Education institution members of the the Computing Research and Education Association of Australasia, CORE. CORE is an association of university departments of Computer Science in Australia and New Zealand.

The program committee integrated more than 40 highly regarded academics around the globe including Brazil, Canada, Denmark, France, Germany, Japan, Mexico, Singapore and the US.

Following an international call for papers, we received 165 abstracts and 120 full papers. Each paper was peer-reviewed in full by at least two independent reviewers, and in some cases three or four referees produced independent reviews. The program committee was impressed by the quality of the submissions. Only 35 papers were accepted. This means an acceptance rate of less than 30%. This is again slightly less than the number of papers in previous years, with a more severe acceptance rate. A conscious decision was made to select papers for which all reviews were positive and favorable. Although the Program Committee made careful quantitative and qualitative assessments on the feedback from reviewers, it is remarkable that all those accepted papers had a weighted score of 5 or above. (In a scale from 1 to 7, here

7 corresponds to Strong Accept [award quality],
6 corresponds to Accept [I will argue for this paper] and
5 corresponds to Weak Accept [I vote for this paper, but won't object reject]).

While this challenged the determinations, and some high quality work may not have been included, we are confident that the result is a very solid program and there is a strong contribution in each of the papers reproduced in these proceedings.

Professor Chris R. Johnson from the University of Utah delivered the keynote address titled *Biomedical Computing and Visualization.* Prof. Johnson's research interests are in the area of scientific computing. Particular interests include inverse and imaging problems, adaptive methods, problem solving environments, numerical analysis, biomedical computing, and scientific visualization. This is very appropriate for the diverse links between disciplines in Computer Science and the links to other sciences that the conference is stimulating in this edition.

Based on the Guidelines on Research Practice in Computer Science by CORE, the conference awards a best student paper award to a student author(s)/coauthor(s) provided that:

1. all other non-student co-authors confirm to the program chair(s) that the nominated author has had a substantial participation into the paper and
2. the student academic supervisor confirm to the program chair that the contribution reflected in the paper is the result of a major component from progress for a research higher degree.

The best student paper was awarded to Scott Vallance from Flinders University for his paper *Rendering Multi-Perspective Images with Trilinear Projection* The best paper was awarded to Hayden Melton and Ewan Tempero, both from the University of Auckland, for their paper titled *Identifying Refactoring Opportunites by Identifying Dependency Cycles.*

We would like to thank Prof. John Roddick for his assistance in the production of the proceedings and Kim Taylor for her help with the Conference Management System. We would also like to thank Prof. Jenny Edwards for her support as president of CORE. Last but not least, we would like to thank the chair of the local organizing committee Prof. Young J. Choi and his committee for their efforts in coordination.

Vladimir Estivill-Castro and Gillian Dobbie
ACSC 2006 Program Chairs
January, 2006

Programme Committee

Chairs

Vladimir Estivill-Castro, Griffith University, Queensland, Australia
Gillian Dobbie, University of Auckland, New Zealand

Members

Hussein A. Abbass, UNSW@ADFA, Australia
Michael H. Albert, University of Otago, New Zealand
Karin Becker, Pontificia Universidade Catolica do Rio Grande do Sul, Brazil
Andrew Bernat, Computing Research Association, USA
Stephane Bressan, National University of Singapore, Singapore
Fred Brown, University of Adelaide, Australia
Neville Churcher, University of Canterbury, NewZealand
Dominique Decouchant, C.N.R.S., France
Trevor Dix, Monash University, Australia
Gill Dobbie, University of Auckland, New Zealand
Jenny Edwards, University of Technology, Sydney, Australia
Vladimir Estivill-Castro, Griffith University, Australia
Mark Evered, University of New England, Australia
Mike Fellows, University of Newcastle, Australia
Colin Fidge, Queensland University of Technology, Australia
Ken Hawick, Massey University - Albany,, New Zealand
Annika Hinze, Waikato University, New Zealand
Jan Hoffmann, Humboldt-Universitüt zu Berlin, Germany
Nigel Horspool, University of Victoria, Canada
Michael Houle, National Institute for Informatics, Japan
Chris Johnson, Australian National University, Australia
Jyrki Katajainen, University of Copenhagen, Denmark
Paddy Krishnan, Bond University, Australia
Xuemin Lin, University of New South Wales, Australia
Bruce Litow, James Cook University, Australia
Bernard Mans, Macquarie University, Australia
Chris McDonald, University of Western Australia, Australia
Mirka Miller, University of Ballarat, Australia
Kara L. Nance, University of Alaska Fairbanks, USA
Philip Ogunbona, University of Wollongong, Australia
Michael Oudshoorn, Montana State University, USA
Alfredo Sanchez, Universidad de las Americas-Puebla, Mexico
Jin Song Dong, National University of Singapore, Singapore
Leon Sterling, University of Melbourne, Australia
Masahiro Takatsuka, University of Sydney, Australia
Bruce H. Thomas, University of South Australia, Australia
Andrew Turpin, RMIT, Australia
Alexandra Louise Uitdenbogerd, RMIT, Australia
Hua Wang, University of Southern Queensland, Australia
Geoff West, Curtin University of Technology, Australia
Graham Williams, Australian Taxation Office, Australia
Burkhard Wuensche, University of Auckland, New Zealand
Yanchun Zhang, Victoria University, Australia
Wanlei Zhou, Deakin University, Australia
Roger Zimmermann University of Southern California, LA, USA

Additional Referees

Bernhard Aichernig
Brad Alexander
Saeed Araban
Sakire Arslan
Eduardo Augusto Bezerra
Jennifer Badham
Ian Barnes
Cail Borrell
Ole Borup
Rhodes Brown
Ney Laert Vilar Calazans
Phil Cook
Hai Dam
Graham Farr
Yuzhang Feng
Jacob de Fine Skibsted
Tarik Hadzic
Morten Halkjer
Jörgen Havsberg Seland
Nicolas Henschel
Claus Jensen
Craig Jones
Andrei Josephsen
Ed Kazmierczak
Mehrdad Khodai-Joopari
Carlo Kopp
Vladik Kreinovich
Susan K. Land
Geoff Leach
Yuan Fang Li
Leslie S. Liu
Qing Liu
Beth Loga
Stephan Lynge
Keith MacKenzie Frampton
Petra Malik
Robi Malik
Teddy Mantoro
Kim Marshall
Aloys Mbala
Tim McComb
Sule Nair
Lee Naish
Andrew Paplinski
Ian Peake
Jovan Pehcevski
David Pereira
Asad Pirzada
Stephen Seidman
Jialie Shen
Anthony Sloane
Sie Teng Soh
Jon Sporring
Bala Srinivasan
Linda Stern
Phil Stocks
Jun Sun
Kuldar Taveter
Chris Thorne
Phil Vines
Haojun Wang
Ang Yang
Yidong Yuan
Xiuzhen Zhang
Hong Zhu
Uwe R. Zimmer

Organising Committee

Welcome

On behalf of the Tasmanian Organising Committee of ACSW2006 I would like to welcome all the delegates to the conferences of this busy and interesting week, in particular those coming from overseas.

The location of the various conferences and other events at the Wrest Point Hotel allows delegates to move quickly from event to event, and to easily and comfortably gather in groups for those conversations and interactions that are so important for the exchange of ideas and the promotion of cooperation, not to mention social pleasure.

We trust you will have a thoroughly enjoyable time.

Professor Young Ju Choi
Chair, Organising Committee
January, 2006

General Chair

Professor Young Ju Choi, School of Computing, University of Tasmania, Australia

Organising Committee Members

Ms Nicole Clark
Dr Julian Dermoudy
Mr Tony Gray
Mr Neville Holmes
Mr Ian McMahon
Ms Julia Mollison
Professor Arthur Sale
Ms Soon-ja Yeom

CORE - Computing Research and Education

CORE welcomes all delegates to ACSW2006 in Hobart.

ACSW, the Australasian Computer Science Week continues to grow with new conferences becoming entrenched in the week. As the premier annual Computer Science event in Australia and New Zealand, it provides an unparalleled opportunity for the wide community of Computer Science academics and researchers to meet, network, promote IT research and be exposed to the latest research in other areas of IT. The research presented at each conference is of the highest standard and essential for the growth and future of our region, in an ever more competitive world.

CORE is expanding its awards. The Distinguished Service Award first offered in late 2004 will be offered every second year and next at the 2007 Conference. Along with the Chris Wallace Research Award, we are offering an annual teaching award for the first time.

CORE has continued to play a part in the Federation of Australian Scientific and Technological Societies and by participating in events such as Science Meets Parliament, CORE is becoming recognised by the wider community and will continue to do so. A major contribution from many members in 2005 was a submission to the RQF Forum with some of our ideas appearing in the draft. CORE and members of the Executive have also been interviewed as representatives of the Computer Science community for several other Government and industry inquiries and initiatives.

Thank you all for your contributions in 2005 and we look forward to an exciting 2006.

Jenny Edwards
President, Computing Research and Education
January, 2006

ACSW Conferences and the Australian Computer Science Communications

The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as *Volume 0*). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

2008. Communications Volume Number 30. Proposed Host and Venue - University of Wollongong, NSW.
2007. Volume 29. Host and Venue - University of Ballarat, VIC.
2006. Volume 28. Host and Venue - University of Tasmania, TAS.
2005. Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.
2004. Volume 26. Host and Venue - University of Otago, Dunedin, New Zealand. First running of APCCM.
2003. Volume 25. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Adelaide Convention Centre, Adelaide, SA. First running of APBC. Incorporation of ACE. ACSAC held separately from 2003.
2002. Volume 24. Host and Venue - Monash University, Melbourne, VIC.
2001. Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.
2000. Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUIC.
1999. Volume 21. Host and Venue - University of Auckland, New Zealand.
1998. Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.
1997. Volume 19. Hosts - Macquarie University and University of Technology, Sydney. Venue - Sydney, NSW. ADC held with DASFAA (rather than ACSW) in 1997.
1996. Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.
1995. Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.
1994. Volume 16. Host and Venue - University of Canterbury, Christchurch, New Zealand. CATS run for the first time separately in Sydney.
1993. Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.
1992. Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).
1991. Volume 13. Host and Venue - University of New South Wales, NSW.
1990. Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).
1989. Volume 11. Host and Venue - University of Wollongong, NSW.
1988. Volume 10. Host and Venue - University of Queensland, QLD.
1987. Volume 9. Host and Venue - Deakin University, VIC.
1986. Volume 8. Host and Venue - Australian National University, Canberra, ACT.
1985. Volume 7. Hosts - University of Melbourne and Monash University. Venue - Melbourne, VIC.
1984. Volume 6. Host and Venue - University of Adelaide, SA.
1983. Volume 5. Host and Venue - University of Sydney, NSW.
1982. Volume 4. Host and Venue - University of Western Australia, WA.
1981. Volume 3. Host and Venue - University of Queensland, QLD.
1980. Volume 2. Host and Venue - Australian National University, Canberra, ACT.
1979. Volume 1. Host and Venue - University of Tasmania, TAS.
1978. Volume 0. Host and Venue - University of New South Wales, NSW.

Conference Acronyms

ACE. Australian/Australasian Conference on Computing Education.
ACSAC. Asia-Pacific Computer Systems Architecture Conference (previously Australian Computer Architecture Conference (ACAC).
ACSC. Australian/Australasian Computer Science Conference.
ACSW. Australian/Australasian Computer Science Week.
ADC. Australian/Australasian Database Conference.
APBC. Asia-Pacific Bioinformatics Conference.
APCCM. Asia-Pacific Conference on Conceptual Modelling.
AUIC. Australian/Australasian User Interface Conference.
CATS. Computing - The Australian/Australasian Theory Symposium.

Note that various name changes have occurred, most notably the change of the names of conferences to reflect a wider geographical area.

ACSW and ACSC 2006 Sponsors

We wish to thank the following sponsors for their contribution towards this conference. For an up-to-date overview of sponsors of ACSW 2006 and ACSC 2006, please see `http://www.comp.utas.edu.au/acsw06/`.

UTAS

University of Tasmania, Australia

AUSTRALIAN COMPUTER SOCIETY

Australian Computer Society

COmputing Research & Education

CORE - Computing Research and Education

Griffith UNIVERSITY

Griffith University, Australia

THE UNIVERSITY OF AUCKLAND NEW ZEALAND

University of Auckland, New Zealand

Keynote Paper

Biomedical Computing and Visualization

Chris R. Johnson and David M. Weinstein

Scientific Computing and Imaging Institute
School of Computing
University of Utah
50 S. Central Campus Drive, Salt Lake City, UT 84112, US

crj@sci.utah.edu, dmw@sci.utah.edu

Abstract

Computers have changed the way we live, work, and even recreate. Now, they are transforming how we think about and treat human disease. In particular, advanced techniques in biomedical computing, imaging, and visualization are changing the face of biology and medicine in both research and clinical practice. The goals of biomedical computing, imaging and visualization are multifaceted. While some images and visualizations facilitate diagnosis, others help physicians plan surgery. Biomedical simulations can help to acquire a better understanding of human physiology. Still other biomedical computing and visualization techniques are used for medical training. Within biomedical research, new computational technologies allow us to "see" into and understand our bodies with unprecedented depth and detail. As a result of these advances, biomedical computing and visualization will help produce exciting new biomedical scientific discoveries and clinical treatments. In this paper, we give an overview of the computational science pipeline for an application in neuroscience and present associated research results in medical imaging, modeling, simulation, and visualization.[1]

Keywords: Biomedical computing, imaging, problem solving environment, visualization.

1 Introduction

The next decade will see an explosion in the use and the scope of biomedical computing and visualization. Advanced, multimodal imaging and visualization techniques, along with new computational methods, will change the way many biomedical researchers and clinicians do their work. The combination of biomedical imaging and visualization with biomedical simulations will produce information about anatomical structure that is linked to functional data, in the form of electric and magnetic fields, mechanical motion, and biochemistry, and genetics. Such an integrated approach will provide comprehensive views of the human body in progressively greater depth and detail. However, such integration will require significant advances in biomedical computing software infrastructures and corresponding advances in multi-scale biomedical computing, imaging, and visualization algorithms (Johnson 2004).

Over the past two decades, the techniques of computer simulation and visualization have had a substantial impact on the field of biomedicine, as they have on other areas of science and engineering. Computer simulation allows biomedical researchers to subject increasingly sophisticated quantitative and qualitative conceptual models of biological behavior to rigorous quantitative simulation and analysis.

2 Neuroscience Application: Neural Source Imaging

Many times each second, the brain sends electrical impulses racing through the body's web of nerve cells to the motor neurons, where they initiate the electrochemical reactions that cause muscles to contract. Several decades ago, scientists recognized that these excitation currents produce an electrical field that can be detected as small voltages on the scalp. In 1924, German psychiatrist Hans Berger recorded the first electroencelphogram (EEG). The EEG electrode measures the small electrical activity from the brain and contains continuous trains of activity. The practice through which one can infer the inter-cranial sources that give rise to these measurements is termed the *neural source imaging* or *inverse EEG* problem. Neural source imaging is a fundamental problem in neuroscience. Learning precisely which regions of the brain are active at a particular time is a central problem in fields ranging from cognitive science to neuropathology to surgical planning.

While the modern technologies of electrode design and electronic recording apparatus differ significantly from their predecessors, the EEG waveforms are essentially the same as those recorded by Berger. Even with the substantial advances in EEG technology, most of the machines in clinical use today provide relatively coarse descriptions of the overall electrical activity of the heart or brain. This limitation in resolution is primarily due to the fact that standard EEG measurements represent the cumulative electrical activity of the brain as a very small number of simple point sources of bioelectricity. Physicians uses these glimpses to help spot disorders by comparing the patient's EEGs with an atlas of waveforms that correspond to particular disease states. Compressing all this information into a small number of features is very efficient, but can lack the sensitivity and spatial resolution required for diagnosing many illnesses.

In some difficult cases, physicians turn to other techniques that are more invasive, costly, and painful and in rare cases, to exploratory surgery. In some cases of epilepsy, for example, physicians must establish whether the source of this abnormal electrical activity is well localized, and hence operable. At present, this diagnosis may require the application of electrodes directly to the surface of the brain.

Using computer modeling, imaging, simulation and visualization, we are developing diagnostic tools that may reduce the need for these cases of preoperative surgery, by simulating and visualizing the electric fields emanating from the brain. Using large-scale, three-dimensional computer models of the head and brain, we can produce more detailed visual representations of the electrical activity within the brain than the currently used brain snapshots from standard EEGs. A primary goal is to develop these techniques based on painless, risk-free voltage measurements from the head surface and gain information that is now primarily available through highly invasive diagnostic procedures.

3 Computational Science Pipeline

In order to solve the neural source imaging problem from above, we must perform several steps that involve elements of what we call the computational science pipeline: experimental data acquisition (patient image acquisition), mathematical modeling (physical equations that describe bioelectric fields), geometric modeling (segmentation, mesh generation), material modeling (electrical conductivity and diffusion tensor), numerical approximation (large-scale parallel finite element analysis, linear solvers, nonlinear optimization), visualization (of the geometric model, material model, and solutions), and validation (of the models and solutions).

Figure 1 schematically illustrates the “Inverse EEG Pipeline” we have constructed for efficient and interactive neural source imaging. In addition to creating efficient algorithms for each task, it is also important to create useful, integrated software, such as the SCIRun (Parker 1997, Weinstein 2005) software system described in the next section. We now briefly describe the stages within the inverse EEG pipeline.

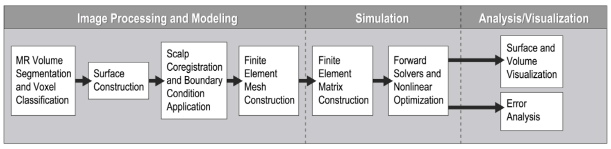


Image Processing and Modeling
Simulation
Analysis/Visualization
MR Volume Segmentation and Voxel Classification
Surface Construction
Scalp Coregistration and Boundary Condition Application
Finite Element Mesh Construction
Finite Element Matrix Construction
Forward Solvers and Nonlinear Optimization
Surface and Volume Visualization
Error Analysis

Fig 1: The Inverse EEG Computational Pipeline.

3.1 MRI Volume Segmentation and Voxel Classification

Our pipeline takes raw MRI data from a scan of a patient's cranium as anatomic input. This stage of the pipeline is accomplished using modules from the Insight Toolkit (ITK) (Yoo 2002) within SCIRun, using, for example a level set algorithm (Lefohn 2003), as shown in Figure 2. The output from this process is a tagged volume of voxels, each labeled with a tag to identify the primary material contained within that voxel. For our application, we are specifically interested in: air, skin, bone, cerebro-spinal-fluid, grey matter and white matter.

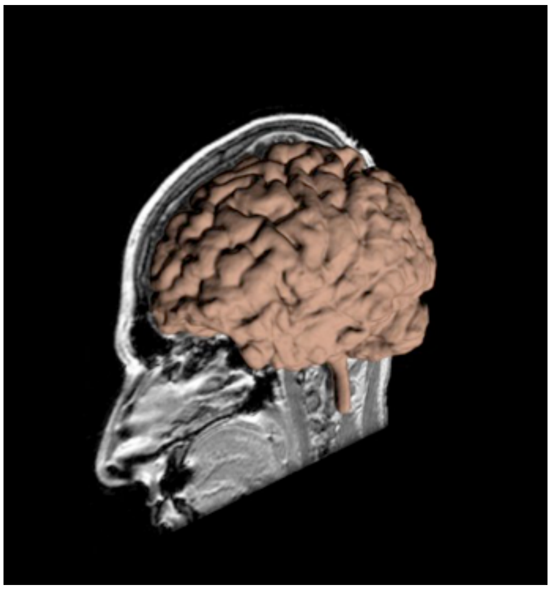


Figure 2: Result of segmentation of the brain using a level set algorithm.

3.2 Surface Construction

From the classified voxels we extract the set of boundary surfaces via a flood-fill/seed-growing style algorithm. Each boundary then corresponds to a discrete material region. Unfortunately, because the segmentation process can leave some noise in the data, we often have on the order of 10,000 surfaces after this process is completed, with many of these surfaces only bounding a single voxel. To reduce the number of surfaces, we pre-process the segmented volume with an assimilation algorithm that annexes regions containing less than some threshold number of voxels into the largest neighboring region. This process has the positive effect of reducing the complexity of the model (where fewer surfaces implies lower complexity), but it can also be destructive if the threshold is set too high. As a result, this was one of the parameters we studied in this study. The surfaces that result from this extraction have a characteristic “staircase” jaginess, since they are composed of voxel faces. To smooth out this data into more physiologically correct (and numerically stable) surfaces, we apply a scanline surface algorithm (Weinstein 2000) to these non-manifold surfaces. The result is shown in the first frame of Figure 3.

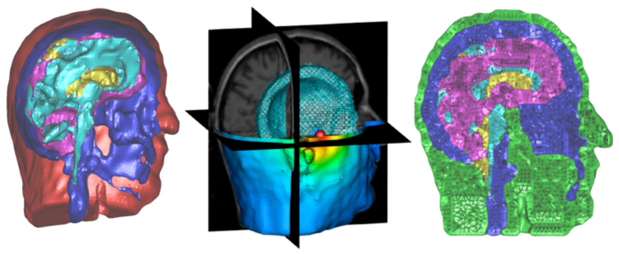


Figure 3: The result of surface construction and mesh generation from segmented MRI data.

3.3 Scalp Co-registration and Boundary Condition Application

In addition to the raw MRI data, we use two other clinically obtained raw datasets in our pipeline. These datasets are the basis for the functional data that will be mapped onto the scalp surface and serve as Dirichlet boundary conditions. The first of these datasets are the potentials recorded through electrodes attached to the patient's scalp. The second dataset is a list of point locations in space, obtained with a pointing device and a magnetic tracker. These points are used to spatially locate the electrode positions on the MR dataset, and consist of electrode positions and a cloud of points digitized off the patient's scalp. The first step in the co-registration process is to match the point cloud to the scalp surface extracted in the surface construction stage. This is done with a semi-automatic algorithm that applies affine transformations to the point cloud to minimize the summed squared distances from the points in the point cloud to the surface. The second step of this process is to apply the boundary conditions to the scalp surface. We simply determine the closest scalp point to each of the electrodes and assign it the corresponding potential with a Dirichlet boundary condition.

3.4 Finite Element Mesh Construction

To construct our finite element mesh, we use a spatial subdivision algorithm that subdivides space into uniform cubic voxels and places a single mesh node in each voxel. Voxels that correspond to air are not included in this process. The placement of each node is chosen based on two criteria: if a surface passes through the voxel, the node is constrained to lie on the surface; nodes must maintain a minimal distance from each other (a Poisson disk constraint, applied between neighboring voxels is used to guarantee this property). The edge lengths of these cubic voxels will directly determine the number of nodes generated. Since fewer nodes will result in less accurate meshes geometrically, we varied this parameter to evaluate the effect of geometric inaccuracies on the cortical solution. After generating all of the nodes, we use the CAMAL mesh generator (Sandia 2004) to construct a tetrahedral mesh. Each element in the mesh is tagged with a material/conductivity.

3.5 Finite element matrix construction

The finite element matrix is constructed by discretizing a generalized Poisson equation, $\nabla \cdot \sigma \nabla \Phi = -I_V$, where Φ is the voltage, σ is the electrical conductivity tensor, and I_V is the electric current per unit volume. The details of the finite element theory and implementation are described in detail in (Johnson 1997). The result of this algorithm is a sparse, symmetric, positive-definite stiffness matrix that encodes all of the geometry and electrical conductivity information of the problem.

3.6 Nonlinear Optimization

In order to solve the source localization problem, one needs to use nonlinear optimization. This involves solving the discretized Poisson equation above multiple times in order to find the global minimum of a misfit function defined as the difference between the measured voltages on the surface of the scalp and the computed solutions assuming a model neural source. We used both a multi-restart simplex search and a simulated annealing algorithm to find the global minimum of the misfit function. Both algorithms recovered the same neural sources, modeled as dipoles. The simplex search algorithm was restarted eight times for each source in order to improve the likelihood that we had localized the global minimum. We validated our recovered minima through an exhaustive search of the domain. Details of the algorithm and implementation can be found in (Weinstein 2000).

3.7 Visualization

Researchers at the SCI Institute and collaborators have created several novel visualization techniques to visualize scalar, vector, and tensor fields (Kniss 2005, Livnat 2005, Scheuermann 2005, Whitaker 2005, Zhang 2005). As an example of a new multi-field visualization technique, we applied a combination of stream surface visualization with simple tensor field visualization to look at the effects of including anisotropy within a realistic head model for the EEG source localization simulation. Figures 4 and 5 illustrate the visualization of the effects of white matter anisotropy using these techniques. We can observe a correlation between the primary direction of the conductivity structure of the white matter fiber bundles and the direction of the return currents. The visualization of return currents in bioelectric field problems can reveal important details about the distribution of sources, interactions at conductivity boundaries, and the effect of geometric distortion on bioelectric fields. By integrating the stream surfaces with a visualization of the diffusion tensors representing the white matter, we can better understand the structural, spatial relationships (Wolters 2005).

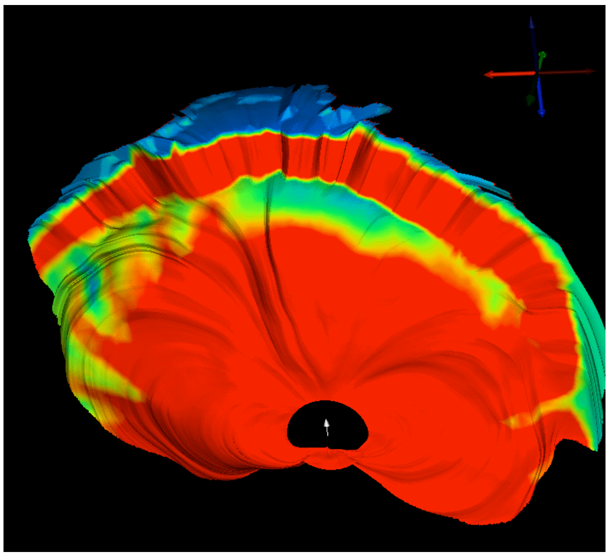


Figure 4: Visualization of return current surfaces from an EEG simulation using an isotropic conductivity model.

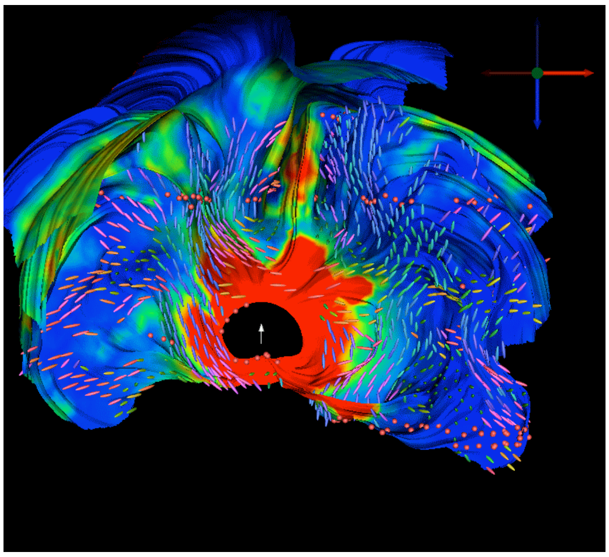


Figure 5: Visualization of return current surfaces from an EEG simulation using an anisotropic conductivity model.

In Figure 6 we show the orientation of the anisotropic white matter tracks from a diffusion tensor MR scan using a novel application of super quadric glyphs (Kindlmann 2004).

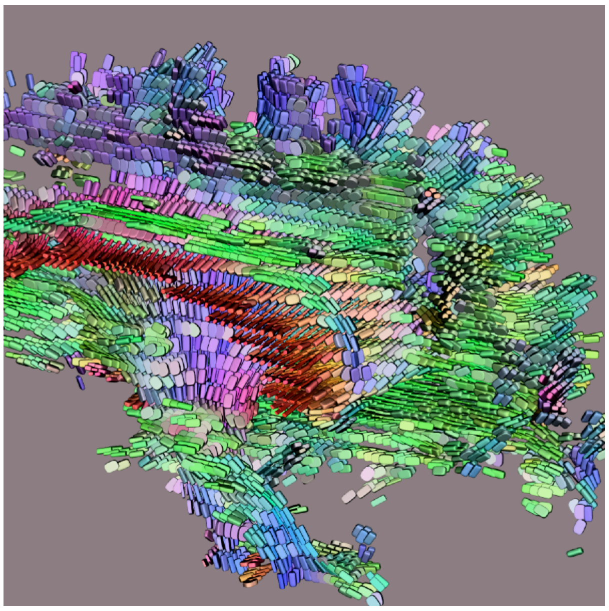


Figure 6: Visualization of half a brain DT-MRI volume using super quadric glyphs. Red indicates left/right, green indicates anterior/posterior, and blue indicates superior/inferior.

4 SCIRun: Integrated Software System

The desire to understand biological systems drives researchers to create ever-more sophisticated computational models. While such sophistication is essential to good research, the resulting complexity of the scientific computing process has itself become a major hindrance to further progress. Sources of this complexity include the number of equations and variables required to encapsulate realistic function, the size of the resulting systems and data sets, and the diverse range of computational resources (algorithms, databases, software, and hardware) required to support significant advances. Biomedical computing researchers gather multi-channel and multi-modal data from real-time collection instruments, access large distributed databases, and rely on sophisticated simulation and visualization systems for exploring biomedical systems.

Managing such large-scale computations requires powerful hardware and efficient and transparent software that frees the user to engage the complexity of the scientific problem rather than of the tools themselves. Unfortunately, such biomedical computing software does not currently exist. The range of computational tools available is growing so rapidly that navigating this large set of possible options has become its own challenge. The need to integrate software is especially acute when scientists seek to create models that span spatial or temporal scales or cross physical systems (e.g. combining electrical with mechanical and biochemical parameters). Integration is also necessary across the various components of the modeling and simulation process. No single researcher has the skills required to master all the computational and biological knowledge needed to successfully create geometric and mathematical models, map them to numerical algorithms, implement them efficiently in modern computers, visualize the results, and understand them as they pertain to the specific biological system under investigation. To successfully model such complex systems requires a multidisciplinary team of specialists, each with complementary expertise and an appreciation of the interdisciplinary aspects of the system, and each supported by a software infrastructure that can leverage specific expertise from multiple domains and integrate the results into a complete software system.

Problem-solving environments (PSEs)[2] provide a natural platform to support integration and leverage multidisciplinary expertise to create complete systems for biomedical computing (Bramley 2000). Such systems solve the challenges of interfacing disparate elements and provide a level of functional abstraction that greatly assists researchers dealing with complex software systems.

PSEs also provide infrastructure for vertical integration of computational knowledge. Specific elements that may be incorporated into a comprehensive PSE include knowledge of the relevant discipline(s); the best computational techniques, algorithms and data structures; the associated programming techniques; the relevant user interface and human-computer interface design principles; the applicable visualization and imaging techniques; and methods for mapping the computations to various computer architectures (Bramley 2000). A PSE can consolidate knowledge from a range of experts in these disparate areas into a system that offers the end user a powerful set of computational tools.

Within the Scientific Computing and Imaging (SCI) Institute at the University of Utah, we have a long history of research in software architecture and creating problem-solving environments for scientific computing, such as SCIRun, BioPSE, and Uintah (SCIRun 2005).

The SCIRun PSE allows the interactive creation, investigation, and steering of large-scale scientific computations. SCIRun has been under development since the mid 1990s, but it has been enhanced significantly over the past five years due to the efforts of two large research centers that have used SCIRun as their core software system. These centers are 1) The Center for the Simulation of Accidental Fires and Explosions (C-SAFE), a Department of Energy ASHLI ASAP Level 1 Center; and 2) the NIH NCRR Center for Integrative Biomedical Computing. Largely because of these efforts, SCIRun has become a comprehensive software environment for scientific computing applications. SCIRun provides a component model, based on a generalized dataflow paradigm, which allows different computational components and visualization components to be connected in a tightly integrated fashion. A dataflow model implies the following: 1) data is sent to a software component, 2) the component manipulates the data in some manner, and 3) the new data is sent downstream to the next component for further manipulation.

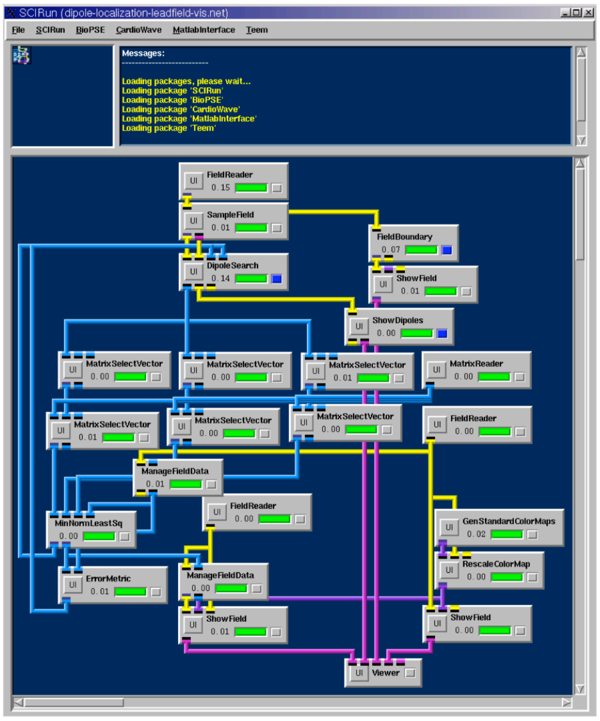
SCIRun can be viewed as a *computational workbench*, in which a scientist designs and modifies a simulation interactively via a component-based visual programming model. SCIRun also facilitates interactive debugging and steering of large-scale, typically parallel, scientific simulations by, for example, enabling a scientist to modify geometric models and interactively change numerical parameters and boundary conditions. As opposed to the typical off-line simulation mode - in which the scientist manually sets input parameters, then computes results, and finally visualizes the results via a separate visualization package, and then starts again at the beginning - SCIRun closes the loop, combining each of these phases of the scientific investigation of the chosen problem.

While SCIRun provides the framework and software support needed to provide the extensive functionality discussed above, the actual science is done by individual software components. The modules are stand-alone pieces of software designed by various individuals or groups and contributed to the system. It is through combining the functionality of a number of modules that interesting problems are solved.

SCIRun/BioPSE Example of EEG Simulation and Visualization

An example electroencephalography (EEG) neural source localization application is show in Figures 7 and 8. Figure 7 contains the dataflow network that implements an inverse EEG application. At the top of the network, the input data files are loaded; these include the finite element mesh that defines the geometry and conductivity properties of the model and a precomputed lead-field matrix that encodes the relationship between electric sources in the domain and the resulting potentials that would be measured at the electrodes. Further down in the network, we have a set of modules that optimize the dipole location in order to minimize the misfit between the measured potentials from the electrodes and the simulated potentials due to the dipole. Finally, we have visualization and rendering modules, which provide interactive feedback to the user.

[2] We note that there are a number alternative phrases for what we mean by a problem solving environment currently being used in the scientific software literature, including software frameworks, toolkits, scientific software environments, software workbenches, plus a number of application specific names.



```
Messages:
-------------------------

Loading packages, please wait...
Loading package 'SCIRun'
Loading package 'BioPSE'
Loading package 'CardioWave'
Loading package 'MatlabInterface'
Loading package 'Teem'
```

Figure 7: SCIRun/BioPSE modules combined for EEG modeling (unstructured mesh generation), simulation (finite element simulation, parallel linear system solves, and inverse source localization), and visualization (mesh visualization, isosurface extraction, and vector field visualization.

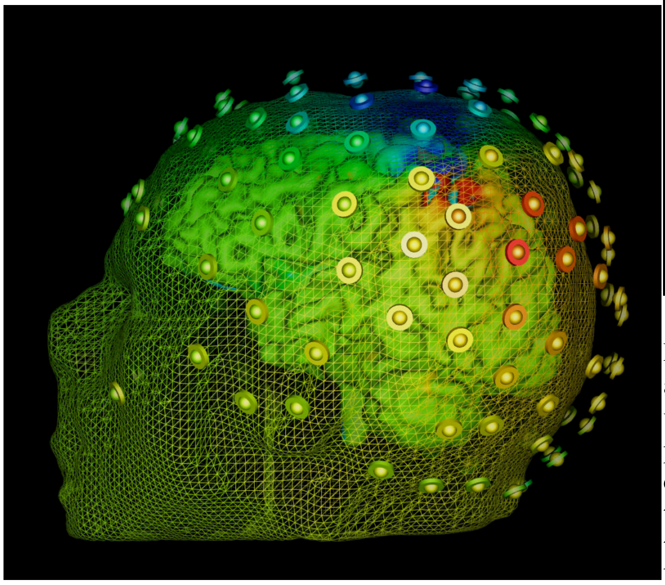


Figure 8: Visualization of simulation results of an EEG simulation localizing a neural source.

PowerApps

One of the major hurdles to SCIRun becoming a practical tool for the scientists and engineers has been SCIRun's dataflow interface. While visual programming is natural for computer scientists and some engineers, who are accustomed to writing software and building algorithmic pipelines, it is overly cumbersome for application scientists[3]. Even when a dataflow network implements a specific application (such as the bioelectric field simulation network provided with BioPSE and detailed in the BioPSE Tutorial), the user interface (UI) components of the network are presented to the user in separate UI windows, without any semantic context for their settings. For example, SCIRun provides file browser user interfaces for reading in data. However, on the dataflow network all of the file browsers have the same generic presentation. Historically, there has not been a way to present the filename entries in their semantic context, for example to indicate that one entry should identify the electrodes input file and another should identify the finite element mesh file.

While this interface shortcoming has long been identified, it has only recently been addressed. We recently introduced *PowerApps*. A PowerApp is a customized interface built atop a dataflow application network. The dataflow network controls the execution and synchronization of the modules that comprise the application, but the generic user interface windows are replaced with entries that are placed in the context of a single application-specific interface window. Figure 9 shows the BioFEM PowerApp implementation of the neural source localization application.

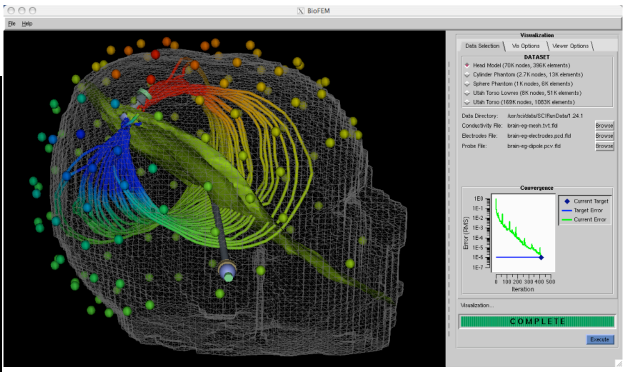


Figure 9: The BioFEM custom interface. Though the application is functionality equivalent to the dataflow version shown in Figure 7, this PowerApp version provides an easier-to-use custom interface. Everything is contained within a single window; the user is lead through the steps of loading and visualizing the data with the tabs on the right; and generic control settings have been replaced with contextually appropriate labels; and application-specific tooltips (not shown) appear when the user places the cursor over any user interface element.

[3] We note this statement is often true of software written by computer science researchers being used by application scientists and engineers.

5 Next Generation Software Architecture: SCIRun2

At the SCI Institute, we are beginning development of a

next-generation software architecture, called SCIRun2 (Zhang 2004). This system shares much of its software code-base with SCIRun, and it is our intent to evolve SCIRun into SCIRun2 over the next year.

SCIRun2 seeks to remove barriers to software component reuse by employing a flexible component architecture that enables a number of different styles of components (called component models) to be used together simultaneously. Thus far, we have been very successful in writing component wrappers to allow software packages (such as ITK, Teem, MATLAB, the CAMAL mesh generator, and so forth) to be used as modules in SCIRun. All of these undertakings have been successes: they have broadened the applicability of SCIRun, improved its performance, and have made it a more useful tool for our collaborators and for the scientific community at large. In practice, though, some of these efforts were very straightforward while others required significant custom development to overcome the technical hurdles.

SCIRun2, born of our experience developing SCIRun, provides a new internal architecture that is specifically designed to integrate component-based and object-based software such as the libraries described above, making this task of integration both simpler and more powerful.

The primary innovative design feature of SCIRun2 is a meta-component model that facilitates integration of a number of classes of tools from various, previously incompatible systems. In the same way that components plug into a traditional component-based PSE (such as the original SCIRun), SCIRun2 will allow entire component models to be incorporated dynamically. SCIRun2 facilitates the coupling of multiple component models, each of which can bring together a variety of components. In addition, the SCIRun2 architecture directly enables features that we wish to add to SCIRun, such as support for MPI-based components, a separation of the user interface from the computational engine, improved scripting support, and features for collaboration.

6 Acknowledgments

This work was supported, in part, by a grant from the NIH NCRR 5P41-1RR12553-07 and from grants from DARPA, DOE, and NSF. SCIRun, BioPSE, and PowerApps software are all available as Open Source from the SCI Institute website (www.sci.utah.edu). SCIRun, BioPSE and the PowerApps are currently supported on three different platforms: Linux, Macintosh OSX, and SGI IRIX. A Windows port will be available in early 2006.

7 Summary

Advanced biomedical computing techniques coupled with advances in multi-modal imaging and visualization will change the way many biomedical researchers and clinicians do their work. The combination of biomedical imaging, and visualization with biomedical simulations will produce information about anatomical structure that is linked to functional data, in the form of electric and magnetic fields, mechanical motion, and biochemistry, and genetics. Such an integrated approach will provide comprehensive views of the human body in progressively greater depth and detail. However, such integration will require significant advances in biomedical computing software infrastructures and corresponding advances in multi-scale biomedical computing, imaging, and visualization algorithms.

8 References

Johnson, C.R., MacLeod, R.S., Parker, S.G., and Weinstein, D.M. (2004): Biomedical Computing and Visualization Software Environments. *Communications of the ACM*, **47** (11): 64-71.

Parker, S.G., Weinstein, D.M., and Johnson, C.R (1997): The SCIRun Computational Steering Software System. In *Modern Software Tools in Scientific Computing*, 1-40. Arge, E., Bruaset, A.M. and Langtangen, H.P. (eds). Birkhauser Press.

Weinstein, D.M., Parker, D.M., Simpson, J., Zimmerman, K., and Jones, G. (2005): Visualization in the SCIRun Problem-Solving Environment. In *The Visualization Handbook*, 615-632. Hansen, C.D. and Johnson, C.R. (eds). Elsevier.

Yoo, T.S., Ackerman, M.J., Lorensen, W.E, Schroeder, W., Chalana, V. Aylward, S., Metaxes, D., and Whitaker, R. (2002): Engineering and Algorithm Design for an Image Processing API: A Technical Report on ITK - The Insight Toolkit. In *Proc. of Medicine Meets Virtual Reality*, (586-592). Westwood, J. (ed). IOS Press Amsterdam.

Lefohn, A.E., Cates, J.E., and Whitaker, R.T (2003): Interactive, GPU-Based Level Sets for 3D Segmentation. In *Medical Image Computing and Computer Assisted Intervention* (MICCAI), 564-572.

Weinstein, D.M. (2000): Scanline Surfacing: Building Separating Surfaces from Planar Contours. In *Proceeding of IEEE Visualization 2000*, 283-289.

Sandia National Laboratories (2004): CAMAL - The CUBIT Adaptive Meshing Algorithm Library - http://cubit.sandia.gov/camal.html - Release 2.0.2.

Johnson, C.R. (1997): Computational and Numerical Methods for Bioelectric Field Problems. In *Critical Reviews in BioMedical Engineering*, **25**(1):1-81.

Weinstein, D.M., Zhukov, L. and Johnson, C.R. (2000): Lead-Field Bases for EEG Source Imaging. *Annals of Biomedical Engineering,* 28(9):1059-1065.

Wolters, C.H., Anwander, A., Tricoche, X, Lew, S., and Johnson, C.R. (2005): Influence of Local and Remote White Matter Conductivity Anisotropy for a Thalamic Source on EEG/MEG Field and Return Current

Computation. In *International Journal of Bioelectromagnetism* (In Press).

Kindlmann, G. (2004): Superquadric Tensor Glyphs. In *Proceeding of The Joint Eurographics - IEEE TCVG Symposium on Visualization 2004*, (147-154).

Kniss, J.M., Kindlmann, G., Hansen, C.H. (2005): Multidimentional Transfer Functions for Volume Rendering. In *The Visualization Handbook*, (189-210). Hansen, C.D. and Johnson, C.R. (eds). Elsevier.

Livnat, Y. (2005): Accelerated Isosurface Extraction Approaches. In *The Visualization Handbook*, (39-55). Hansen, C.D. and Johnson, C.R. (eds). Elsevier.

Scheuermann, G. and Tricoche, X (2005): Topological Methods for Flow Visualization. In *The Visualization Handbook*, (341-356). Hansen, C.D. and Johnson, C.R. (eds). Elsevier.

Whitaker, R.T. (2005): Isosurfaces and Level-Sets. In *The Visualization Handbook*, (97-123). Hansen, C.D. and Johnson, C.R. (eds). Elsevier

Zhang, S., Laidlaw, D.H., and Kindlmann, G. (2005): Diffusion Tensor MRI Visualization. In *The Visualization Handbook*, (327-340). Hansen, C.D. and Johnson, C.R. (eds). Elsevier

Bramley, R., Char B., Gannon, D. , Hewett, T., Johnson, C.R., and Rice, J. (2000): Enabling Technologies for Computational Science: Frameworks, Middleware, and Environments. In *Workshop on Scientific Knowledge, Information, and Computing*, (19-32). Houstis, E., Rice, J., Gallopoulos, E, and Bramley, R. (eds). Kluwer Academic.

SCIRun, BioPSE, and PowerApp Software. Scientific Computing and Imaging Institute. http://www.sci.utah.edu/.

Zhang, K., Damevski, K., Venkatachalapathy, V., Parker, S. (2004): SCIRun2: A CCA Framework for High Performance Computing, In *Proceedings of The 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments*.

Full Papers

Logic and Refinement for Charts

Greg Reeve **Steve Reeves**

Department of Computer Science,
University of Waikato,
New Zealand,
Email: {gregr,stever}@cs.waikato.ac.nz

Abstract

We introduce a logic for reasoning about and constructing refinements for μ-Charts, a rational simplification and reconstruction of Statecharts. The method of derivation of the logic is that a semantics for the language is constructed in Z and the existing logic and refinement calculus of Z is then used to induce the logic and refinement calculus of μ-Charts, proceeding by a series of definitions and conservative extensions and hence generating a sound logic for μ-Charts, given that the soundness of the Z logic has already been established.

Keywords: Statecharts, reactive systems, Z, $\mathcal{Z_C}$, logic, refinement

1 Introduction

The specification language μ-Charts is a rational simplification and reconstruction of Statecharts (Harel 1987). As such, it can be considered to define the core of the many Statechart-like languages: a family of visual languages that are used for designing reactive systems. It is simpler than the original Statecharts, the simplification being achieved by omitting some of the more complicated and reportedly less-used constructs. It is designed to have a more comprehensible semantics, without losing expressiveness. One important contribution of this work, then, is a semantics and logic for the core of Statecharts, presented independently of any particular tool or other "operational" embodiment of semantics and logic.

In the past a formal semantics has been given to μ-Charts using both a process algebraic, traces approach and denotationally using automata by Scholz (1998), along with a logical treatment (Reeve & Reeves 2000) using the specification language Z (Spivey 1989),(13568 2002). While different aspects, and versions, of μ-Charts have been published (Philipps & Scholz 1997*a*, Philipps & Scholz 1997*b*, Scholz 1998), the definitive account (prior to the development of the Z-based logic) was published by Scholz (1998). Characteristic features of μ-Charts are that transitions are *instantaneous* (and hence input and output signals appear simultaneously); communication using selected signals (feedback) between charts is *local* (pairwise) rather than global and is defined explicitly; and that charts may *nondeterministically* choose between transitions.

The language semantics assumes a chaotic-outside-of-defined-behaviour interpretation. The language is also distinguished from process algebras where the natural interpretation is often one where transitions are triggered by the presence of required signals. μ-Charts also allows transitions to be triggered by the absence of designated signals. This is facilitated by imagining there is a global clock whose tick causes all transitions leaving the state of a chart to evaluate their guards.

The aim of this paper is to present a partial relations-based logic for μ-Charts and and then to show how a refinement theory can be lifted from Z to charts.

In this paper we view μ-Charts as a language for specification, especially since we allow nondeterminism. In a subsequent paper we will show how implementations can be built: the method will be to use the language and notions of refinement presented here to move our specification towards implementation by gradually reducing nondeterminism and adjusting the interfaces as required, and then using the program development work by, *e.g.*, Henson and Reeves (2003) and the more recently by Henson *et al.* (2004) to arrive at an implementation.

The derivation of the logic rules is somewhat simplified due to space constraints: the interested reader can consult work by Reeve (2005) for a more detailed account. For similar reasons the formal semantics given covers just one of the three language constructs.

We do not give examples of reasoning about any particular chart: how to use a logic to reason is, we assume, second nature to our audience and is straightforward. Rather, we use the logic for the far more ambitious and fundamental goal of deriving refinement rules for charts.

Section 2 presents an introduction to the formal treatment of μ-Chart's semantics. This includes describing one of the language operators, giving the general method of deriving a Z model for that operator and the derivation of natural deduction-style logic rules using the Z logic. We divide this section into two: the first part shows how the semantics for charts is given in Z; the second part shows how the Z semantics is given a meaning and then how rules for charts can be derived.

Section 3 shows how we use the existing and well-investigated refinement notion of Z to derive a refinement notion for charts. This is concluded with a discussion of what this refinement notion is in terms of the more traditional process algebraic, trace description of chart behaviour. In Section 4 we consider monotonicity of refinement in general, and in Section 5 we show how the logic we have developed can be used to investigate and prove monotonicity properties of μ-Charts. Finally, we present some conclusions in Section 6.

2 The μ-Charts Logic

This section provides an introduction to μ-Charts via the definition its of semantics in Z. The logic and semantics of Z itself is then used to induce a logic and set-theoretic semantics for μ-Charts.

This process is divided into two natural phases: we first use definitions to express the semantics of μ-Charts in Z. These definitions define what we call the *transition model*, which is essentially a function which maps expressions from μ-Charts to Z, and which we denote by $[\![\cdot]\!]_{Z_t}$. We then use the standard mapping from Z into the underlying, core language $\mathcal{Z_C}$ (Henson & Reeves 2000). This mapping is denoted by $[\![\cdot]\!]_{\mathcal{Z_C}}$. The composition of these two semantic mappings then gives us the semantics of μ-Charts in $\mathcal{Z_C}$. We then use the logic of $\mathcal{Z_C}$ together with the definitions that go to make up $[\![\cdot]\!]_{Z_t}$ to induce a logic for μ-Charts. Since this logic has been constructed from the sound logic for $\mathcal{Z_C}$ by a series of conservative extensions (via definitions), we know that the logic for μ-Charts is sound too.

A μ-chart is either atomic or a combination of charts using language operators. An atomic chart is essentially a finite-state automaton where a transition in the chart is labelled with a pair, *guard/action*. That the transition is taken (or triggered) is conditional on the guard being satisfied by the current input signals and leads to the action happening, meaning that signals are output as required by that action. Each chart has an input interface designating signals that can trigger a transition and an output interface designating signals that the chart can output. A fundamental assumption is that time passes in the control states of a chart, but the transitions between these states occur *instantaneously*. A chart reaction is therefore characterised by the input of a set of signals from the environment and the instantaneous output of a set of signals to the environment. This reaction is called a step or a tick of the clock, but note that this does not mean that the intervals between reactions must be equal.

A chart C has an input interface in_C that typically includes all of the signals that appear in its transition guards and an output interface out_C that typically includes all of the signals that appear in its transition actions.

In order to define large reactive systems, the language has three structuring mechanisms: parallel composition, hierarchic decomposition and input/output interface definition. In a parallel composition, each component chart reacts *synchronously* on a global clock. A feedback mechanism between pairs of charts makes output signals instantaneously available as input signals, which allows component charts to communicate *asynchronously* on signals. A composition chart $C = C_1 \mid \Psi \mid C_2$, which composes charts C_1 and C_2 with feedback on signals in the set Ψ, has an input interface $in_C = in_{C_1} \cup in_{C_2}$ and an output interface $out_C = out_{C_1} \cup out_{C_2}$.

A further structuring mechanism means some of the states of a chart need not be atomic, but rather one chart may be embedded in another ("inside" one of its states) as a sub-chart, using hierarchic decomposition. Finally, the definition of the assumed context of a chart via the explicit definition of the input and output interfaces of a chart (of arbitrary structure) allows signal hiding.

The full definition of the language semantics, via the logic, treats each of these language operators separately. In this paper we will concentrate on just the definition of atomic charts and the parallel composition operator.

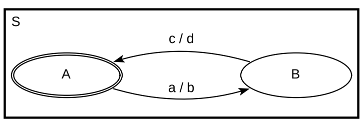


Figure 1: A simple atomic μ-chart

2.1 The transition model

The transition model essentially relates the current configuration of a chart and input to a new configuration and the resulting output. This relation describes every possible step that a chart can take. In contrast, a typical way to give a semantics for such languages is to use sets of observable input/output traces, for example the trace semantics given by Scholz (1998). This abstracts on the control states by defining just its reactions in an assumed context. The link between the logical semantics described here and a *trace* semantics can be constructed by considering a chart making one step after another and recording just the input and resulting output. The resulting trace semantics is considered fully by Reeve (2005).

2.1.1 Atomic charts

An atomic chart has the general textual form (Name, State set, Start state, Feedback signals, transition function). Consider the chart $(S, \{A, B\}, A, \{\}, \delta)$ (where δ is the appropriate transition description) pictured in Figure 1. [1]

Informally the behaviour that this chart captures can be described as: the chart starts in state A; if it is in state A and the signal a is input then signal b is output and the chart changes to state B; and similarly if it is in state B and c is input then d is output and the new state is A.

The essence of the transition model for an atomic μ-chart is the description of each of its transitions using a separate Z operation schema. These operation schemas (one for each transition in the chart) are combined using Z schema disjunction to give one schema that describes the transition behaviour of the chart. The Z state of the model has an observation that indicates the current configuration of the chart. The operation schemas describe each transition, that is, how and when that configuration changes.

For an atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$ we introduce Z axiomatic definitions (Figure 2) that model the set of possible chart states, the input interface and the output interface. The sets μ_{State} and $\mu Signal$ contain all allowable state and signal symbols.

The state schema $Chart_C$ records the current state of the chart C using observation c_C. The initial state of the chart is modelled by the schema $Init_C$.

A separate state schema called $C\sigma$ is given for each state in the chart $\sigma \in \Sigma$.

Next we give an operation schema for each chart transition. That is, for each transition $(S_f, S_t, guard/action) \in \delta$ we define an operation schema named $\delta_{S_f S_t}$.

We can see from this definition that each binding in the set has five observations. The meanings of these are:

- c_S—the state of the chart before the transition happens, in this case the state A

[1] We use the piece of text which is the name of the chart to refer to both the chart itself and its name, allowing the context to indicate which we mean. The fact that the name stands for itself as a piece of text is used in the syntactic process of defining the Z that a chart has as its semantics, as we shall see.

$$\begin{array}{|l}
states_C : \mathbb{P}\,\mu_{State} \\
in_C : \mathbb{P}\,\mu Signal \\
out_C : \mathbb{P}\,\mu Signal \\
\Psi : \mathbb{P}\,\mu Signal
\end{array}$$

$$Chart_C \;==\; [c_C : states_C]$$

$$\begin{array}{|l}
\hline
\textit{— } Init_C \\
Chart_C \\
\hline
c_C = \sigma_0 \\
\hline
\end{array}$$

$$C\sigma \;==\; [Chart_C \mid c_C = \sigma]$$

$$\begin{array}{|l}
\hline
\textit{— } \delta_{S_f S_t} \\
CS_f \\
CS'_t \\
i_C : \mathbb{P}\, in_C \\
act : \mathbb{P}\,\mu_{State} \\
o'_C : \mathbb{P}\, out_C \\
\hline
C \in act \\
\rho(guard) \\
o'_C = action \\
\hline
\end{array}$$

Figure 2: The Z semantics, the transition model, for an atomic chart

- i_S—the set of input signals which are offered by the environment that are in the input interface of the chart
- act—a set that denotes all currently active charts
- c'_S—the state of the chart after the transition happens, in this case B
- o'_S—the output generated by the chart, in this case the set containing the signal b

Note that part of the definition of atomic charts—the observation act—is part of the mechanism that allows for the definition of the hierarchic decomposition operator. As we will not be looking at the details of this operator in this paper, it is sufficient to understand that a chart can be active or inactive, and transitions happen only when an atomic chart is active, which is recorded by having its name contained in act. Hence the predicate $C \in act$ is part of the precondition of the operation schema $\delta_{S_f S_t}$.

The predicate $\rho(guard)$, introduced in schema $\delta_{S_f S_t}$, stands for the Z predicate that models the syntactic guard of a chart transition. If we consider a transition's guard in general as a (possibly empty) list of signal expressions, separated by the conjunction symbol &, then each of the elements in the list can be classified into two categories: either a positive signal expression—simply the name of a signal; or a negative signal expression—the signal name is prefixed with a minus sign. A positive signal expression, say sig where $sig \in in_C$, is denoted by the Z expression $sig \in i_C \cup (o'_C \cap \Psi)$. A negative signal expression, say $-sig$, is denoted by the Z expression $sig \notin i_C \cup (o'_C \cap \Psi)$. The syntactic construction process denoted by ρ determines the appropriate predicate for each signal expression and connects them together using the Z logical conjunction operator $\wedge$. If the list is empty the predicate (produced by the process ρ) would be $true$. So, for the transition labelled a/b in chart S in Fig.1 the predicate produced would be $a \in i_c \cup (o'_c \cap \Psi)$ since the guard of this transition is just a.

This general scheme for giving the Z for a transition defines the semantic function $[\![.]\!]_{Z_t}$. For an arbitrary transition $(S_f, S_t, guard/action)$ we have,

$$[\![(S_f, S_t, guard/action)]\!]_{Z_t} \;=_{def}\; \delta_{S_f S_t}$$

The schema $\delta_{S_f S_t}$ provides the Z semantics for the transition.

Along with the schemas for each transition we also need a single schema that models the behaviour of the chart when it is inactive (Figure 3). Again this Z is part of the general transition necessary to model the entire language, in particular, including the decomposition operator. Here it is enough to realise that an inactive chart plays no part in output. For the general atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$, we name the inactive schema $Inactive_C$.

Now, the entire transition model for an atomic chart is given by Definition 2.1[2]

Definition 2.1

$$[\![(C, \Sigma, \sigma_0, \Psi, \delta)]\!]_{Z_t} \;=_{def}\; (\bigvee\{[\![t]\!]_{Z_t} \mid t \in \delta\}) \vee Inactive_C$$

in a context containing the axiomatic definitions and $Chart_C$, $Init_C$ and $C\sigma$.

The complete transition model for chart S from Figure 1 is defined as the disjunction of each of the individual transition schemas, *i.e.* $\delta_S == \delta_{AB} \vee \delta_{BA} \vee Inactive_S$, and two of these schemas are given as examples in Figure 4.

2.1.2 The composition operator

The composition operator allows us to take two μ-charts C_1 and C_2 and join them together to form a new, more complex chart $C_1 \mid \Psi \mid C_2$ where Ψ is a set of signals on which C_1 and C_2 can communicate. As mentioned, the charts run separately but synchronously, *i.e.* in lock step with one another. Their only medium of communication is asynchronous via the multicast of those signals in the set Ψ. The communication is asynchronous in that output is always enabled—a chart can always broadcast signals. However, there is no guarantee that the other chart in the composition is listening, that is, ready to react on the signals broadcast. Signals persist only during one step of the chart.

The transition model $[\![C]\!]_{Z_t}$ for the composed chart $C = C_1 \mid \Psi \mid C_2$ contains a similar set of Z definitions and schemas (Figure 5) as that for an atomic chart. Here it is assumed that any entity subscripted with C_1 comes from the transition model of the chart C_1 and similarly for C_2.

2.1.3 Partial relations semantics

The step semantics of a chart is no more than the transition model of the entire μ-chart specification

[2]We use the notation $\bigvee X$ to denote the schema disjunction of all the schemas in the set X.

$$\begin{array}{|l}
\hline
\textit{— } Inactive_C \\
\Xi Chart_C \\
i_C : \mathbb{P}\, in_C \\
act : \mathbb{P}\,\mu_{State} \\
o'_C : \mathbb{P}\, out_C \\
\hline
C \notin act \\
o'_C = \{\} \\
\hline
\end{array}$$

Figure 3: Z semantics for Inactive

$$\begin{array}{|l}
\multicolumn{1}{l}{\delta_{AB}} \\
SA \\
SB' \\
i_S : \mathbb{P}\, in_S \\
act : \mathbb{P}\, \mu_{State} \\
o'_S : \mathbb{P}\, out_S \\
\hline
S \in act \\
a \in i_S \cup (o'_S \cap \{\}) \\
o'_S = \{b\} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{Inactive_S} \\
\Xi Chart_S \\
i_S : \mathbb{P}\, in_S \\
act : \mathbb{P}\, \mu_{State} \\
o'_S : \mathbb{P}\, out_S \\
\hline
S \notin act \\
o'_S = \{\} \\
\hline
\end{array}$$

Figure 4: Z for the chart in Figure 1

with the active state machinery hidden. Given an arbitrary μ-chart called C, the step behaviour of C is defined by another schema which, by convention, we call $CSys$ (Figure 6).

The schema $CSys$ (right) hides the active state observation and specifies that the top-most chart of any hierarchical structure is active.

So, $[\![\,]\!]_{Z_t}$ for an arbitrary chart C generates various pieces of Z, depending on the structure of C, defining the transition model. We then have to give a meaning to the Z in order to generate logical rules, which we now do.

2.2 The $\mathcal{Z_C}$ model

From the transition model of a chart as given above we move on to give a logic for charts by modelling the transition model in $\mathcal{Z_C}$ hence deriving introduction and elimination rules that allow us to prove properties about a chart's transition model and hence about a chart.

2.2.1 Atomic charts

Given the general method for constructing the Z semantics of a chart (*i.e.* the transition model), we can describe the meaning of the chart by describing the meaning of the Z. To do so we rely on a reasonable level of familiarity with the meta-language used in the presentation of the kernel logic $\mathcal{Z_C}$ in Henson and Reeves (2000). Briefly, we: use the binding concatenation operator $\star$; restricted membership $\dot{\in}$; restricted equality $\doteq$; type meta-variables for example T; αT as shorthand for all observations of the schema type T; superscript type meta-variables to denote the types of bindings and schemas; and the type union operator $\curlyvee$. For brevity, we suppress mention of types (indicated by superscripts on terms) in all cases where this is possible and rely on the unique of types that $\mathcal{Z_C}$ enjoys to assure ourselves of the well-formedness (which includes well-typedness) of our terms.

Returning to the example chart of Figure 1, again, the Z meaning of the transition from configuration A to B is given by the schema δ_{AB}, whose meaning in turn is given in the theory $\mathcal{Z}_C$ as a set of bindings as follows:

$$\begin{array}{l}
[\![\delta_{AB}]\!]_{\mathcal{Z}_C} =_{def} \\
\{ \langle\!| \; c_S \Rrightarrow A,\; i_S \Rrightarrow i,\; act \Rrightarrow active,\; c'_S \Rrightarrow B,\; o'_S \Rrightarrow \{b\} \; |\!\rangle \mid \\
\qquad i \subseteq in_S \wedge active \subseteq \mu_{State} \bullet S \in active \wedge a \in i \}
\end{array}$$

$$\begin{array}{|l}
states_C : \mathbb{P}\, \mu_{State} \\
in_C : \mathbb{P}\, \mu Signal \\
out_C : \mathbb{P}\, \mu Signal \\
\Psi : \mathbb{P}\, \mu Signal \\
\hline
states_C = \\
\qquad states_{C_1} \cup states_{C_2} \\
in_C = in_{C_1} \cup in_{C_2} \\
out_C = out_{C_1} \cup out_{C_2}
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{Chart_C} \\
Chart_{C_1} \\
Chart_{C_2} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{Init_C} \\
Init_{C_1} \\
Init_{C_2} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\delta_C} \\
\Delta Chart_C \\
i_C : \mathbb{P}\, in_C \\
act : \mathbb{P}\, \mu_{State} \\
o'_C : \mathbb{P}\, out_C \\
\hline
C_1 \in act \Leftrightarrow C \in act \\
C_2 \in act \Leftrightarrow C \in act \\
\exists\, i_{C_1}, i_{C_2}, o_{C_1}{}', o_{C_2}{}' : \mathbb{P}\, \mu Signal \bullet \\
\qquad i_{C_1} = (i_C \cup (o'_C \cap \Psi)) \cap in_{C_1} \wedge \\
\qquad i_{C_2} = (i_C \cup (o'_C \cap \Psi)) \cap in_{C_2} \wedge \\
\qquad o'_C = o_{C_1}{}' \cup o_{C_2}{}' \wedge \delta_{C_1} \wedge \delta_{C_2} \\
\hline
\end{array}$$

Figure 5: Semantics for composition

$$\begin{array}{|l}
\multicolumn{1}{l}{CSys} \\
\Delta Chart_C \\
i_C : \mathbb{P}\, in_C \\
o'_C : \mathbb{P}\, out_C \\
\hline
\exists\, act : \mathbb{P}\, \mu_{State} \bullet C \in act \wedge \delta_C \\
\hline
\end{array}$$

Figure 6: Top-level semantics

The second schema $Inactive_S$ describes the behaviour of the chart when it is inactive, and its meaning is given as:

$$\begin{array}{l}
[\![Inactive_S]\!]_{\mathcal{Z}_C} =_{def} \\
\{ \langle\!| \; c_S \Rrightarrow s,\; i_S \Rrightarrow i,\; act \Rrightarrow active,\; c'_S \Rrightarrow s,\; o'_S \Rrightarrow \{\} \; |\!\rangle \mid \\
\qquad s \in \{A, B\} \wedge active \subseteq \mu_{State} \wedge i \subseteq in_S \bullet S \notin active \}
\end{array}$$

By definition of schema disjunction the set $[\![\delta_S]\!]_{\mathcal{Z}_C}$, which gives in $\mathcal{Z_C}$ the meaning of the transition model for the chart, contains all of the bindings from the sets $[\![\delta_{AB}]\!]_{\mathcal{Z}_C}$, $[\![\delta_{BA}]\!]_{\mathcal{Z}_C}$ and $[\![Inactive_S]\!]_{\mathcal{Z}_C}$.

In order to make the logical rules we are working towards slightly more readable, we define the predicate *Trans* which captures what it means for a binding of the transition model to characterise a transition of the chart. Given an arbitrary transition of the form $t = (S_f, S_t, guard/action)$, from the chart C, we have,

$$\begin{array}{l}
Trans\ t\ z^T =_{def} \\
\qquad z.c_C = t.S_f \wedge \rho(t.guard)[\alpha T / z.\alpha T] \\
\qquad \wedge\, z.c'_C = t.S_t \wedge z.o'_C = t.action
\end{array}$$

The terms $t.S_f$ *etc.* are assumed to be defined in the obvious way such that $t.S_f$ gives the "from state" of a transition, $t.guard$ gives the guard component of a transition, $t.S_t$ gives the "to state" and $t.action$

gives the action component. ρ is as defined above and constructs a predicate from a guard.

Now we give the formal definition of the transition model for charts directly in terms of the meaning of the Z model.

Definition 2.2 For the arbitrary atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$, we have:

$$\begin{aligned}
&[\![\delta_C]\!]_{\mathcal{Z}_C} =_{def} \\
&\quad \{z \mid C \notin z.act \wedge z \mathrel{\dot{\in}} \Xi Chart_C \wedge z.o'_C = \{\} \ \vee \\
&\qquad\qquad C \in z.act \wedge \exists\, t \in \delta \bullet Trans\ t\ z\}
\end{aligned}$$

From this definition we finally derive introduction and elimination rules.

Proposition 2.1 Given the atomic chart $(C, \Sigma, \sigma_0, \Psi, \delta)$, where for arbitrary binding z we have:

$$\frac{z \mathrel{\dot{\in}} \delta_C \quad actv\ C\ z \quad t \in \delta, Trans\ t\ z \vdash P}{P}\ ({Z_t}^{-})$$

$$\frac{actv\ C\ z \quad t \in \delta \quad Trans\ t\ z}{z \mathrel{\dot{\in}} \delta_C}\ ({Z_t}^{+})$$

assuming the usual conditions (due to the elimination of an existential quantifier) for t and P, and where, for an atomic chart C, the predicates $actv\ C\ z$ and $inactv\ C\ z$ are defined as follows:

$$actv\ C\ z =_{def} C \in z.act$$

$$inactv\ C\ z =_{def} \neg\ actv\ C\ z$$

2.2.2 Composition

We give the definition of the $\mathcal{Z}_{\mathcal{C}}$ model for composed charts in terms of the meaning of the transition model in:

Definition 2.3 Given an arbitrary composition $C = C_1 \mid \Psi \mid C_2$ we have,

$$\begin{aligned}
&[\![\delta_C]\!]_{\mathcal{Z}_C} =_{def} \\
&\{z \mid C_1 \in z.act \Leftrightarrow C \in z.act \wedge \\
&\qquad C_2 \in z.act \Leftrightarrow C \in z.act \wedge \\
&\qquad \exists\, o_1, o_2 \bullet z.o'_C = o_1 \cup o_2 \wedge \\
&z \star \langle\!|\ i_{C_1} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_1}, {o_{C_1}}' \Rrightarrow o_1\ |\!\rangle \mathrel{\dot{\in}} \delta_{C_1} \wedge \\
&z \star \langle\!|\ i_{C_2} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_2}, {o_{C_2}}' \Rrightarrow o_2\ |\!\rangle \mathrel{\dot{\in}} \delta_{C_2}\}
\end{aligned}$$

The introduction and elimination rules for composed charts shown in Figure 7 are derived from this definition.

2.2.3 The step semantics

Now Definition 2.4 defines the step semantics for a chart.

Definition 2.4 For arbitrary chart C with the associated Z description $CSys$,

$$\begin{aligned}
&[\![CSys]\!]_{\mathcal{Z}_C} =_{def} \\
&\qquad \{z \mid \exists\, z_1 \bullet\ z_1 \doteq z \wedge actv\ C\ z_1 \wedge z_1 \in \delta_C\}
\end{aligned}$$

From this definition we derive introduction and elimination rules given in Figure 8.

We often refer to the step semantics as the *partial relations semantics*. This is because the meaning of the schema $CSys$ can be considered as a relation that maps the "before" configuration of a chart and input to its "after" configuration and output. This relation is often partial because a μ-Chart specification describes the reaction to some input events and not others.

3 The μ-Charts Refinement Calculus

In Derrick and Boiten (2001) and Woodcock and Davies (1996), a framework for considering Z specifications and Z refinement in terms of abstract data types (ADTs) is introduced. The idea is to map a "standard" Z specification, *i.e.* state schema, initialisation schema and operation schemas, into a relational ADT setting. Broadly a relational ADT is a tuple of the form (X, xi, xf, Ops) such that: X is a state space; xi is an initialisation relation; xf is a corresponding finalisation relation; and Ops is an indexed set of relational operations. The initialisation and finalisation relations map a global observable state into the ADT's private state and vice versa. A program of an ADT is defined as a particular sequence of the indexed operations upon a data type, preceded by initialisation and ended by finalisation. This mapping is used to derive a data refinement theory for Z specifications from the existing refinement notion for partial relations ADTs.

Given that the partial relation semantics for μ-Charts is defined via Z we can fit charts into the same framework. Recall from Section 2 that the Z model of a chart constitutes a state space, an initialisation schema and one operation schema, this operation schema being the description of every step that the chart can take. If we view the Z model of a chart in the ADT framework we can say that any program allowed by the chart is an example of composing the step operation together with itself again and again. Of course, what we are really interested in is the sequences of inputs and outputs that result from such programs. If we imagine running this program over all possible input sequences and recording the resulting output sequences then we have exactly the trace semantics of the chart. Because we are modelling reactive systems we choose to consider the traces over infinite sequences of input and output. Therefore, we need to imagine composing the step operation with itself indefinitely.

In the following we show how we can generalise the Z/ADT results to charts. In particular we show that the ADT view of a chart can be considered as giving the trace semantics of that chart. Then we derive a notion of partial relations refinement for charts based on an existing notion of partial relation refinement for Z.

We diverge from what may be considered the usual way to give a refinement notion in a reactive systems setting, that is, using the *behavioural approach* (as Derrick and Boiten (2001) calls it) for completing partial relations, and assume chaotic behaviour outside of the preconditions of partial relations (which is the more common-to-Z notion too, as it happens). The resulting notion of refinement is particularly interesting because it allows us to refine both the behaviour of a reactive system and the context, via the system's interface, in which we assume that reactive system will reside. This notion of refining both behaviour and context is not new to this work. The notion of "chaotic refinement" for *specifications* of reactive systems was suggested in the original definition of the language μ-Charts (Scholz 1998). Of course, since a behavioural interpretation is available in the Z framework, we could also derive more traditional rules for a reactive system if we wished to, as we typically would when we move from a specification to an implementation.

3.1 Charts and ADTs

The usual account of ADT refinement makes the simplifying assumption that the types of inputs and outputs associated with the abstract and concrete pro-

Proposition 2.2 Given $C = C_1 \mid \Psi \mid C_2$, for the binding z and arbitrary sets o_3 and o_4, we have:

$$\frac{z \mathrel{\dot{\in}} \delta_C \qquad \begin{array}{l} z.o'_C = o_1 \cup o_2, \\ z \star \langle\!| \, i_{C_1} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_1}, o_{C_1}{}' \Rrightarrow o_1 \, |\!\rangle \mathrel{\dot{\in}} \delta_{C_1}, \\ z \star \langle\!| \, i_{C_2} \Rrightarrow (z.i_C \cup z.o'_C \cap \Psi) \cap in_{C_2}, o_{C_2}{}' \Rrightarrow o_2 \, |\!\rangle \mathrel{\dot{\in}} \delta_{C_2}, \\ actv\ C\ z \vee inactv\ C\ z \qquad\qquad \vdash Q \end{array}}{Q} \ (|\_|^{-})$$

where the usual conditions, due to the elimination of existential quantifiers, hold between o_1, o_2 and Q. The predicates $actv\ C\ z$ and $inactv\ C\ z$ are defined for the composed chart $C = C_1 \mid \Psi \mid C_2$ as:

$$actv\ C\ z =_{def} actv\ C_1\ z \wedge actv\ C_2\ z \wedge C \in z.act$$

$$inactv\ C\ z =_{def} inactv\ C_1\ z \wedge inactv\ C_2\ z \wedge C \notin z.act$$

Figure 7: Rules for composition

Proposition 2.3 For arbitrary chart C and binding z we have,

$$\frac{z \mathrel{\dot{\in}} CSys \quad z \star z_a \mathrel{\dot{\in}} \delta_C,\ actv\ C\ z_a \vdash Q}{Q} \ (Z_s^{-}) \qquad \frac{\exists\, y \bullet z \star y \mathrel{\dot{\in}} \delta_C \wedge actv\ C\ y}{z \mathrel{\dot{\in}} CSys} \ (Z_s^{+})$$

where the usual conditions (due to the elimination of an existential quantifier) hold between z_a and Q.

Figure 8: Rules for a step of the system

grams (the two n-operation programs P_a^n and P_c^n) are the same. For our purposes this simplifying assumption is too strict. We allow, under what turn out to be rather strong provisos, the input and output interface of a chart to be changed via refinement. Weakening the assumption of equivalent typed input and output for both abstract and concrete programs is achieved using the respective initialisation and finalisation relations in conjunction with the notion of an observable context for charts. We assume that refinement is a judgement made in the broadest input/output context.

We use the respective initialisation and finalisation relations to make the ADT's global state model the appropriate input/output context. The observable behaviour of the ADT (*i.e.* the global state) is given by the input and output sequences that range over the signals of both charts. The initialisation relation maps the global input sequences into appropriate input sequences for the respective charts. Similarly, the finalisation relation maps the outputs from the respective charts into the global output sequences.

From this we make a link between the semantics for a chart C given by embedding the chart in an ADT framework, denoted by $[\![C]\!]_d^{r\omega}$, and an infinite trace semantic definition of charts, *i.e.* $[\![C]\!]_x^{\omega}$, as follows:[3]

Definition 3.1 For arbitrary chart C and sequences $i \in \mathcal{I}^\omega$ and $o \in \mathcal{O}^\omega$,[4]

$$(i, o) \in [\![C]\!]_d^{r\omega} =_{def} (i_{\rhd(in_C)}, o_{\rhd(out_C)}) \in [\![C]\!]_x^{\omega}$$

Now we follow the well-known relational ADT approach (for example see Woodcock and Davies (1996) and Derrick and Boiten (2001)) to derive refinement rules for charts in terms of their partial relations semantics. Note that Definition 3.1 allows us to relate the resulting refinement notion back into the infinite trace style semantics for charts as given in Scholz (1998).

[3]We assume that $\mathcal{I}_c^*$ denotes the set of finite sequences ranging over the type $\mathbb{P}\,Input$. Similarly, $\mathcal{O}_c^*$ denotes all finite sequences over the type $\mathbb{P}\,Output$. The infinite sequences $\mathcal{I}_c^\omega$ and $\mathcal{O}_c^\omega$ are similarly defined.

[4]The notation $i_{\rhd(in_C)}$ denotes the pointwise restriction of the elements in the sequence i to the elements in the set in_C and similarly for out_C.

3.2 Simulation and Corresponding States

Before we derive the refinement rules we briefly introduce and discuss the concept of simulation. When comparing two charts based on input and output traces, that is, checking for or calculating trace refinements, the state information of the charts is already abstracted away. This is not the case, however, when working with the partial relations semantics. We need a way of relating the states of one chart with those of another. This is exactly the task of simulations, sometimes also known as *retrieve relations*, *abstraction relations*, or *coupling invariants* (Derrick & Boiten 2001). Something as simple as changing the names of the states from the abstract chart to the concrete requires that we have a simulation relation that maps the abstract state names into the new concrete state names.

In the standard ADT treatment, a simulation relation encodes the relationship between the states of the abstract specification and the states of the concrete specification. We usually think of the simulation R as completing a series of commuting squares. This allows us to prove the necessary refinement properties for each of the associated operations (in our case there is only one) and use an inductive argument to show that the refinement holds when we compose (in an appropriate order) several operations together into programs. We refer the reader to Derrick and Boiten (2001) for a detailed description of the concepts of data type refinement.

As discussed in Section 3.1, the initialisation and finalisation relations are used to modify the observable input and output sequences to allow refinement to change the context (*i.e.* input/output interfaces) of a chart. Reflecting this, we split the definition of the simulation relation into two separate parts. The first part is the simulation between configurations of the respective abstract and concrete charts. We will refer to this part of the simulation as the *corresponding relation* or $Corr_C^A$ for a simulation between charts A and C.

The Z schema $Corr_C^A$ (Figure 9) gives the general scheme for the corresponding relation. The predicate P defines the simulation relationship between the states of the respective charts A and C, and will, of course, depend on precisely what charts A and C are.

The second part of the simulation relation allows refinements that change the input/output interfaces

$$\begin{array}{|l}
\multicolumn{1}{l}{Corr^A_C} \\ \hline
Chart_A \\
Chart'_C \\ \hline
P \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{IO^A_C} \\ \hline
i_A : in_A \\
i'_C : in_C \\
o_A : out_A \\
o'_C : out_C \\ \hline
i_A \cap in_C = i'_C \cap in_A \\
o_A \cap out_C = o'_C \cap out_A \\ \hline
\end{array}$$

Figure 9: Semantics for simulation relation

of a chart. For arbitrary input interfaces in_A and in_C, and output interfaces out_A and out_C, the schema IO^A_C is constructed so that $[\![IO^A_C]\!]_{\mathcal{Z}_C}$ represents a relation between the inputs and outputs from the abstract chart to the inputs and outputs of the concrete chart. Importantly, when this relation is combined with the corresponding relation we get a schema representing the simulation relation between charts A and C that has type $\mathbb{P}(T^{io}_A \curlyvee T^{io}_C{}')$ as follows:

Definition 3.2 For charts A and C we have,

$$R^A_C =_{def} Corr^A_C \wedge IO^A_C$$

where $[\![R^A_C]\!]_{\mathcal{Z}_C}$ has type $\mathbb{P}(T^{io}_A \curlyvee T^{io}_C{}')$.

Significantly, when using the refinement theory presented the developer need only define the relationship between states of the "refining" and "refined" charts. The input/output relationship or interface refinement is always constrained by the general relationship identified by the schema IO.

3.3 Partial Relation Refinement

Now we can derive partial relations refinement rules for charts. The derivation of the different sets of rules closely follows a similar treatment by Derrick and Boiten (2001).

We embed the Z-based chart ADT presented so far into a relational data type as follows.

Definition 3.3 For an arbitrary chart C and all sequences si and so, the Z ADT semantics $(Chart_C, Init_C, \{CSys\})$ is embedded in the relational data type $(CState, CInit, \{CStep\}, CFin)$, such that,

$$\begin{aligned}
&CState =_{def} \mathcal{I}^{\omega}_C \times \mathcal{O}^{*}_C \times U_C \\
&CInit =_{def} \{(si \mapsto (si_{\rhd(in_C)}, \langle\rangle, z)) \mid z \in Init_C\} \\
&CFin =_{def} \{(si_{\rhd(in_C)}, so_{\rhd(out_C)}, z) \mapsto so \mid z \in Chart_C\} \\
&CStep =_{def} \{(i \frown si, so, z_1) \mapsto (si, so \frown o, z_2) \mid \\
&\qquad\qquad z_1 \star \langle\!| \; i_C \Rrightarrow i, o'_C \Rrightarrow o \; |\!\rangle \star z'_2 \in CSys\}
\end{aligned}$$

The embedding of the simulation R gives the simulation relation S between the ADTs representing charts A and C.

For arbitrary sequences si and so, and bindings z_1 and z_2 we have,

$$\begin{aligned}
S =_{def} \{&(si_{\rhd(in_A)}, so_{\rhd(out_A)}, z_1) \mapsto \\
&(si_{\rhd(in_C)}, so_{\rhd(out_C)}, z_2) \mid z_1 \star z'_2 \in Corr^A_C\}
\end{aligned}$$

Notice that the relational simulation S is defined just in terms of the corresponding relation $Corr^A_C$. This is because the pointwise restriction of the sequences si and so already model the same relationship between input and output as the schema IO^A_C.

3.4 Total Chaos Refinement

3.4.1 Forward Simulation

Here we derive rules for a total chaotic interpretation of charts. Derrick and Boiten (2001) give five refinement conditions that are necessary to show that a relational data type C refines a relational data type A using a forwards simulation S. They begin by lifting (by introducing the special value $\perp$) and totalising the relations of the respective data types. Derrick and Boiten refer to the total chaotic interpretation as the *contract approach*. After giving the necessary lifted totalised relations they show how the five refinement conditions, referred to as *initialisation*, *finalisation*, *finalisation applicability*, *applicability* and *correctness*, can be simplified ("relaxed") to remove any reference to the introduced value $\perp$. We give the five relaxed conditions and refer to Woodcock and Davies (1996) for their derivations.

Definition 3.4 Assuming data types

$$A = (AState, AInit, \{AStep\}, AFin)$$

and

$$C = (CState, CInit, \{CStep\}, CFin)$$

a forwards simulation S is a relation from $AState$ to $CState$ satisfying the following conditions:

$$\begin{array}{lr}
CInit \subseteq AInit \fatsemi S & \text{(init)} \\
S \fatsemi CFin \subseteq AFin & \text{(fin)} \\
\operatorname{ran}((\operatorname{dom} AFin) \lhd S) \subseteq \operatorname{dom} CFin & \text{(fin app)} \\
\operatorname{ran}((\operatorname{dom} AStep) \lhd S) \subseteq \operatorname{dom} CStep & \text{(app)} \\
((\operatorname{dom} AStep) \lhd S) \fatsemi CStep \subseteq AStep \fatsemi S & \text{(corr)}
\end{array}$$

Now we use each of these conditions along with the relational embedding defined in Definition 3.3 to derive corresponding conditions expressed in Z.

For initialisation we have,

$$\begin{array}{l}
CInit \subseteq AInit \fatsemi S \\
\quad\Leftrightarrow \\
\forall y_c \bullet y_c \mathrel{\dot\in} Init_C \Rightarrow \exists t_1 \bullet t_1 \mathrel{\dot\in} Init_A \wedge t_1 \star y'_c \in R
\end{array}$$

Unlike in the derivation provided by Derrick and Boiten (2001), the finalisation condition does not hold trivially for charts. This difference arises because the derivation for Z refinement makes the assumption that both the abstract and concrete ADTs have equivalently typed input and output, whereas the derivations required here do not.

$$\begin{array}{l}
S \fatsemi CFin \subseteq AFin \\
\quad\Leftrightarrow \\
out_A \subseteq out_C
\end{array}$$

Because the given finalisation relation is total over all output sequences and states of the respective charts, the finalisation applicability condition holds trivially.

Now for the applicability condition we have:

$$\begin{array}{l}
\operatorname{ran}((\operatorname{dom} AStep) \lhd S) \subseteq \operatorname{dom} CStep \\
\quad\Leftrightarrow \\
\forall y_a, y_c \bullet Pre\ ASys\ y_a \wedge y_a \star y'_c \in R \Rightarrow Pre\ CSys\ y_c
\end{array}$$

And finally, for correctness we have:

$$\begin{array}{l}
((\operatorname{dom} AStep) \lhd S) \fatsemi CStep \subseteq AStep \fatsemi S \\
\quad\Leftrightarrow \\
\forall y_a, y_c, z_c \bullet (Pre\ ASys\ y_a \wedge y_a \star y'_c \in R \wedge y_c \star z'_c \mathrel{\dot\in} CSys) \Rightarrow \\
\qquad\qquad \exists t \bullet y_a \star t' \mathrel{\dot\in} ASys \wedge t \star z'_c \in R
\end{array}$$

The completion of these derivations gives us the necessary conditions to show that a relation R is a forwards simulation between two charts A and C under the total chaotic interpretation of the partial relations semantics. As we have shown, it follows that chart C refines A in the total chaotic trace interpretation for charts. In line with the natural deduction style presentation that we have adopted, Figure 10 gives introduction and elimination rules for forwards simulation total chaotic refinement.

Notice the rules for forward simulation refinement presented here are, with the exception of the initialisation and finalisation conditions, very similar to the rules presented by Deutsch and Henson in Deutsch and Henson (2003) for *SF-refinement.* A similar method of derivation gives the corresponding rules for the backwards simulation case.

4 Monotonicity results

As with any language that provides operators allowing modular specifications and a refinement calculus for step-wise development, the monotonicity properties of the μ-Charts operators needs to be considered. These monotonicity properties are important for μ-Charts because they show to what extent the language supports modular development. Refinement is considered monotonic with respect to a language operator if a refinement of one part of a composite specification implies a refinement of the specification as a whole, and having this result is clearly important when we turn to using the logic on large specifications.

It turns out that we need quite strong, but very easy to motivate, side-conditions to guarantee that refinement is monotonic with respect to the chart composition operator.

Even though the monotonicity side-conditions described in Proposition 5.1 are presented before the monotonicity result itself, the conditions were formulated and refined from the proof of the monotonicity property (which we omit here due to space constraints). That the process of proving the monotonicity property allows us to state (and prove) these necessary side-conditions is evidence that the method of this paper has met some important goals. That is, the formal framework presented allows us to formulate precise descriptions of general, and typically non-obvious, language properties. In the case of the monotonicity result presented here, the first of the three required side-conditions is particularly non-obvious and at first reading may appear incorrect. However, the proof of monotonicity and careful evaluation of what this condition actually entails, makes clear the significance of the restriction.

5 Monotonicity of the μ-Charts composition operator

We begin by showing that the composition operator of μ-Charts is monotonic with respect to forward simulation refinement only when appropriate side-conditions hold. Like the investigation of Deutsch *et al.* (2003), the monotonicity proof itself is used to establish the necessary side-conditions. After ascertaining the required side-conditions an intuitive (in chart terms) justification for their necessity is given.

Recall that, by definition 3.4, to show that a forward simulation refinement holds between two charts requires that we show that an appropriate simulation exists between the charts. The proof of monotonicity relies heavily on splitting the definition of the simulation into two parts—the simulation between the respective charts' configurations using the *corresponding relation* and the simulation between the allowable input and output signals using the relation IO. This notion of splitting the simulation relation was introduced in Section 3.2 where we define the corresponding relation between two charts A and C as $Corr^A_C$ and the input/output relation as IO^A_C. Where previously we have denoted (total chaotic) forward simulation refinement between two charts C and A as $C \sqsupseteq_{\tau f} A$, here we supplement the relation with an explicit label that names the simulation required for refinement. So, assuming that chart C refines chart A using the simulation S, we will write $C \sqsupseteq^S_{\tau f} A$.

Proposition 5.1 states the monotonicity result for forward simulation refinement.

Proposition 5.1 If, for arbitrary charts A_1, C_2, and signal set Ψ, we have that,

$$\frac{}{\lceil A_1 \rfloor_\Psi \sqsupseteq^T_{\tau f} \lceil C_2 \rfloor_\Psi}\ SC_1$$

$$\frac{}{out_{A_1} \cap \Psi = out_{C_2} \cap \Psi}\ SC_2$$

$$\frac{}{out_{A_1} \cap out_B = out_{C_2} \cap out_B}\ SC_3$$

where $T =_{def} Corr^{C_2}_{A_1} \wedge IO^{C_\Psi}_{A_\Psi}$ for $C_\Psi = \lceil C_2 \rfloor_\Psi$ and $A_\Psi = \lceil A_1 \rfloor_\Psi$, then for arbitrary chart B, we have the monotonicity result,

$$\frac{C_2 \ \sqsupseteq^R_{\tau f}\ A_1 \quad SC_1 \quad SC_2 \quad SC_3}{(C_2 \mid \Psi \mid B)\ \sqsupseteq^S_{\tau f}\ (A_1 \mid \Psi \mid B)}$$

where $S =_{def} Corr^{A_1}_{C_2} \wedge Corr^B_B \wedge IO^A_C$, and $R =_{def} Corr^{A_1}_{C_2} \wedge IO^{A_1}_{C_2}$.

Despite the intricate appearance of the three side-conditions required for monotonic refinement of composed charts, these conditions are not unexpected when described in terms of charts themselves.

First consider the following property that holds in general for arbitrary charts A_1 and C_2, and feedback set Ψ.

Lemma 5.2

$$\frac{C_2 \ \sqsupseteq^R_{\tau f}\ A_1 \qquad out_{A_1} \cap \Psi = out_{C_2} \cap \Psi}{\lceil C_2 \rfloor_\Psi \sqsupseteq^{T'}_{\tau f} \lceil A_1 \rfloor_\Psi}$$

where $T' =_{def} Corr^{A_1}_{C_2} \wedge IO^{A_\Psi}_{C_\Psi}$

Given this property holds it follows that, in the context of the monotonicity proof of Proposition 5.1, *i.e.* where SC_1 holds, the charts A_1 and C_2 are output equivalent with respect to the signals in the set Ψ, *i.e.* $\lceil C_2 \rfloor_\Psi \approx_{\mathcal{O}} \lceil A_1 \rfloor_\Psi$. In words, an environment that reacts to just those signals in the set Ψ could not tell the difference between the charts A_1 and C_2. Therefore, we see that one of the properties required to guarantee monotonic refinement (with respect to composition) is that refining one part of the composition, say refining chart A_1 into C_2, cannot change the behaviour of A_1 with respect to the signals in Ψ that are used to communicate with the other part of the composition, *e.g.* chart B.

To explain the rôle of the side-condition SC_1 more specifically, with regard to the monotonicity proof, we describe two distinct parts that SC_1 plays in the proof.

Firstly, SC_1 enforces that the precondition of the chart, *i.e.* the set of state/input pairs for which the chart has explicitly defined behaviour, cannot be

Proposition 3.1 For arbitrary charts A and C, and bindings y_a, y_c, and z_c, we have,

$$\frac{\begin{array}{ll} & \vdash out_A \subseteq out_C \\ y_c \,\dot{\in}\, Init_C & \vdash t_1 \,\dot{\in}\, Init_A \\ y_c \,\dot{\in}\, Init_C & \vdash t_1 * y_c' \in R \\ Pre\ \ ASys\ y_a,\ \ y_a \star y_c' \in R & \vdash Pre\ \ CSys\ y_c \\ Pre\ \ ASys\ y_a,\ \ y_a \star y_c' \in R,\ \ y_c \star z_c' \,\dot{\in}\, CSys & \vdash y_a \star t_2' \,\dot{\in}\, ASys \\ Pre\ \ ASys\ y_a,\ \ y_a \star y_c' \in R,\ \ y_c \star z_c' \,\dot{\in}\, CSys & \vdash t_2 \star z_c' \in R \end{array}}{C \sqsupseteq_{\tau f} A} \ (\sqsupseteq^{+}_{\tau f})$$

$$\frac{C \sqsupseteq_{\tau f} A}{out_A \subseteq out_C} \ (\sqsupseteq^{-}_{\tau f\ I}) \qquad \frac{C \sqsupseteq_{\tau f} A \quad y_c \,\dot{\in}\, Init_C \quad t_1 \,\dot{\in}\, Init_A, t_1 \star y_c' \in R \vdash P}{P} \ (\sqsupseteq^{-}_{\tau f\ II})$$

$$\frac{C \sqsupseteq_{\tau f} A \quad Pre\ \ ASys\ y_a \quad y_a \star y_c' \in R}{Pre\ \ CSys\ y_c} \ (\sqsupseteq^{-}_{\tau f\ III})$$

$$\frac{C \sqsupseteq_{\tau f} A \quad Pre\ \ ASys\ y_a \quad y_a \star y_c' \in R \quad y_c \star z_c' \,\dot{\in}\, CSys \quad \begin{array}{r} y_a \star t_2' \,\dot{\in}\, ASys, \\ t_2 \star z_c \in R \vdash P \end{array}}{P} \ (\sqsupseteq^{-}_{\tau f\ IV})$$

where the usual conditions hold, due to elimination of existential quantifiers, between t_1, t_2 and P.

Figure 10: Rules for chaotic refinement

weakened. Note that here we use the term weakening of the precondition in a very strict sense—side-condition SC_1 restricts any weakening of the precondition within the domain defined by the input interface of the abstract specification. Extending the domain of definition for a chart specification, *i.e.* increasing the input interface and weakening the precondition outside of the original domain, is still permitted in general.

This first aspect of the side-condition SC_1 is required for the part of the monotonicity proof related to the *correctness* property introduced in Section 3.4.1.

Figure 11 presents a counter-example that illustrates why this part of side-condition SC_1 is necessary in terms of charts. Given the charts A and C we clearly have that $C_2 \sqsupseteq_{\tau f} A_1$, yet it is **not** the case that $C \sqsupseteq_{\tau f} A$. That is, even though C_2 refines A_1, the composed chart C is not a valid refinement of A. The defined reaction of chart A given input $\{a\}$ is to output $\{w, t\}$, *i.e.* the two left hand transitions of chart A combine with respect to feedback to create an overall chart transition triggered by just the input $\{a\}$. However, chart C can nondeterministically choose to output $\{w, t\}$ or $\{w, s\}$ given input $\{a\}$, *i.e.* both the respective left hand and right hand transitions combine to give this nondeterministic behaviour. Therefore, C has additional nondeterministic behaviour to A and no valid refinement holds.

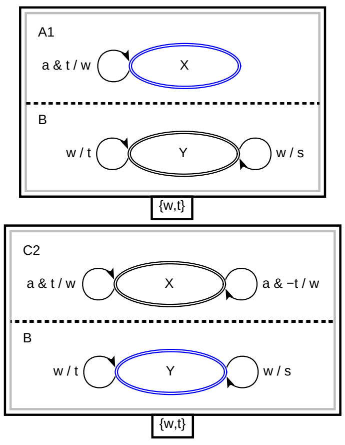


Figure 11: $SC_1, partI$: Charts $A = (A_1 \mid \{w, t\} \mid B)$ and $C = (C_2 \mid \{w, t\} \mid B)$

The second aspect of SC_1 is that it insists that the output behaviour, with respect to feedback, of an abstract specification is not changed via refinement. The property is required to prove the part of the monotonicity result related to the *applicability* condition.

In terms of charts, Figure 12 illustrates another counter-example that demonstrates why this second aspect of SC_1 is a necessary requirement for monotonic refinement. Note that the output interface of the chart C_2 is assumed to contain the signal w, *i.e.* we assume C_2 is a behavioural refinement of A_1 rather than an interface refinement. Again we have that the composed chart C does **not** refine the chart A. This is because A is defined for input $\{a\}$ due to feedback on w where chart C is not. Therefore chart C acts chaotically for input $\{a\}$ and the resulting additional nondeterminism invalidates the refinement relation.

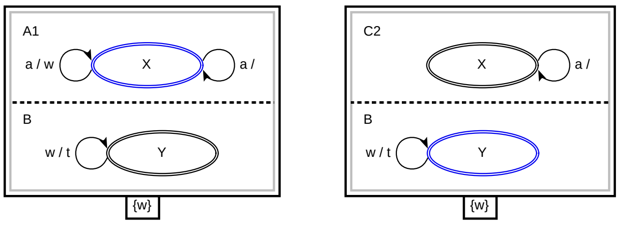


Figure 12: SC_1 and SC_2: Charts $A = (A_1 \mid \{w\} \mid B)$ and $C = (C_2 \mid \{w\} \mid B)$

The same charts from Figure 12 can be used to demonstrate why the side-condition SC_2 is required for monotonicity. In this case, however, we assume that the output interface of chart C_2 is reduced to the empty set of signals, that is, in this case C_2 is an interface refinement of A_1 rather than a behavioural refinement as above. Given this assumption SC_1 holds, that is, $\lfloor A_1 \rfloor_\Psi$ is a valid refinement of $\lfloor C_2 \rfloor_\Psi$. However, from inspection it is obvious that SC_2 does not hold in this case, that is, $out_{A_1} \cap \Psi \neq out_{C_2} \cap \Psi$, specifically, $\{w\} \cap \{w, t\} \neq \{\} \cap \{w, t\}$. The side condition SC_2 is required to prove monotonicity in relation to the *correctness* condition.

Finally, the side-condition SC_3 is required because μ-Charts refinement allows the designer to change the output context of a chart using interface refinement. If an interface refinement of one chart in a composition extends the control that the chart has over the environment using signals that were originally used just by the other part of the composition, then there is the possibility that this new behaviour, from both charts, will be inconsistent when the charts are recombined in composition. For example, consider the counter example illustrated by the charts of Figure 13.

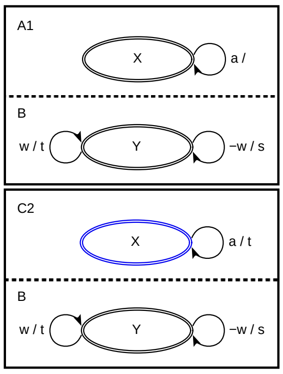


Figure 13: $SC_2(ii)$: Charts $A = (A_1 \parallel B)$ and $C = (C_2 \parallel B)$

Here the valid interface refinement $C_2 \sqsupseteq_{\tau f} A_1$ allows C_2 to control its environment over signals previously dealt with by the chart B, *i.e.* the signal t. The result is that the composed chart C can output $\{s, t\}$ for input $\{a\}$ where chart A could only output $\{s\}$ for input $\{a\}$. Hence, chart C has new behaviour that was not specified by chart A and therefore C is not a valid refinement of A.

Similar arguments can be used to show that the same side-conditions, SC_1, SC_2 and SC_3, are sufficient to guarantee monotonic refinement with respect to the composition operator for charts in the backwards simulation case.

5.1 The firing conditions interpretation of μ-Charts

A requirement for monotonic refinement is that the preconditions remain unchanged over the domain of definition of a chart. This requirement may cause an observant reader to question whether the *total chaotic* and *firing conditions* notions of refinement coincide in the case where refinements adhere to the monotonicity conditions. In particular, the work of Deutsch, Henson and Reeves (2002) shows that refinement based on a *firing conditions* approach can be considered as a notion that insists on the *stability of the precondition*. That is, refinement that allows the reduction of nondeterminism but insists that the precondition is neither strengthened nor weakened.

In fact, we can show that *total chaotic* refinement is both sound and complete with respect to *firing conditions* refinement when we insist that just the first condition SC_1, for monotonic refinement, is met. Any (guaranteed) monotonic refinement that we can prove using the total chaotic rules can also be proved using the rules for firing conditions refinement.

This is expressed by Proposition 5.3 .

Proposition 5.3 For arbitrary charts A, C and signal set Ψ we have,

$$\frac{C \sqsupseteq^{S}_{\tau f} A \qquad \llbracket A \rrbracket_{\Psi} \sqsupseteq^{T}_{\tau f} \llbracket C \rrbracket_{\Psi}}{C \sqsupseteq^{S}_{fcf} A} \qquad\qquad \frac{C \sqsupseteq^{S}_{fcf} A}{C \sqsupseteq^{S}_{\tau f} A}$$

where $S =_{def} Corr^{A}_{C} \wedge IO^{A}_{C}$ and $T =_{def} Corr^{C}_{A} \wedge IO^{C_{\Psi}}_{A_{\Psi}}$ for $C_{\Psi} = \llbracket C \rrbracket_{\Psi}$ and $A_{\Psi} = \llbracket A \rrbracket_{\Psi}$.

Notice that the second aspect of the side-condition SC_1 and the conditions SC_2 and SC_3 are still a necessary requirement to guarantee that firing conditions-based refinements are monotonic with respect to composition.

Therefore, while it is the case that using $\sqsupseteq_{fcf}$ for chart refinement implies a "more monotonic" refinement calculus, the difference in reality is slight.

The exact difference between the two notions of refinement is that the total chaotic model allows a refinement to weaken the precondition over the abstract domain of definition where the firing conditions model does not. The choice of the appropriate model can only be determined by the context of the refinement application. We do point out, though, that the total chaotic model provides the most general refinement framework.

6 Conclusions

A logic for composition and refinement of μ-Charts has been presented. The presented work has two significant contributions. The first is the presentation of a method for developing a logic for a StateCharts-like specification language—another example of the increasingly popular visual specification languages for reactive systems. The second is an investigation of a chaotic-based notion of refinement for the language μ-Charts.

The logic is developed by modelling the μ-Charts language in the more well-known and investigated language of Z. Given the extensive body of work that gives a logic to Z, we can specialise this logic and thereby induce a logic for charts. (Also, we are able to utilise existing tools for Z to reason about the model of a reactive system, if we wish.)

The notion of refinement that we induce for μ-Charts follows the chaotic-outside-of-defined-behaviour approach that is typically associated with Z-based ADT refinement or data refinement. As with Z-based refinement, the chart refinement defined maintains the principle of substitutivity (Derrick & Boiten 2001). That is, the substitution of an implementation of the specification for an implementation of a refinement of the specification will be indistinguishable in the context of the specification.

The notion of a chaotic semantics for μ-Charts was first introduced by Scholz (1998). The implicit nondeterminism outside of defined behaviour can be considered an abstraction mechanism just as in Z specifications. As the design is refined the nondeterminism is reduced, *i.e.* more decisions are made about undefined behaviour.

The chaotic semantics also facilitates refinements of both a reactive system's specified behaviour, and the specified context of the reactive system. These two types of refinement are both defined in the one notion of refinement presented. Refinement that changes the context of a chart preserves substitutivity because it is assumed that the context for a specified chart is fully defined, *i.e.* the context both controls and is controllable by just the signals in the respective input and output interfaces of the specification. Note that hiding signals from one or other of the interfaces is not in general a refinement.

The refinement rules presented give half (*i.e.* the forward simulation case) of a simulation based refinement calculus for μ-Charts. Unlike other theories of refinement for reactive systems the calculus presented allows the simulations to model a change in possible states from abstract to concrete specification as well as a change in the signals used to interact with the environment, *i.e.* the context, of the specified reactive system.

Of course, using the methods of this paper, as much as required of the whole of StateCharts can have a logic induced for it—once a semantics for a given construct has been defined in Z it is an intricate but conceptually straightforward task to induce logical rules for the construct from the $\mathcal{Z}_\mathcal{C}$ rules and the definition. The same goes, also, for refinement rules.

6.1 Future Work

Given the origins of μ-Charts (it is based on a simplification of the more well-known StateCharts), we typically take for granted that μ-Charts is a useful engineering tool for specifying reactive systems. Not surprisingly, to date most of the uses of the presented logic for μ-Charts have been concerned with investigating and proving properties of the language itself. It remains to be shown whether or not such a logic can be used practically to reason about and develop reactive systems. It is clear however, that using the formal logic for the practical development of reactive systems will require significant tool support. Ideally, this would be proof assistance, based specifically on the logic rules for charts, perhaps using a more general tool developed for the logic $\mathcal{Z}_C$. Other Z-based validation tools such as animation may also provide useful tools for investigating μ-Chart specifications. Limited tool support for μ-Charts already exists including a μ-Charts editor called AMuZed and a model builder (*i.e.* a program that translates a chart into its Z model) called ZooM (*Z-lambda project* 2005).

Another application of the logic that has not been fully investigated is to use the form of the simulations involved in refinement to suggest useful refinements of reactive system specifications. That is, can the form of proofs of refinements be used to indicate useful development strategies for reactive systems? This application of the logic closely follows Dijkstra's notion that "we develop program and correctness proof hand-in-hand" (Dijkstra 1976).

References

13568, I. (2002), *Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics*, Prentice-Hall International series in computer science, first edn, ISO/IEC.

Derrick, J. & Boiten, E. (2001), *Refinement in Z and Object-Z: Foundations and Advanced Applications*, Formal Approaches to Computing and Information Technology, Springer.
URL: *http://www.cs.ukc.ac.uk/pubs/2001/1200*

Deutsch, M. & Henson, M. C. (2003), An analysis of forward simulation data refinement, *in* D. Bert, J. Bowen, S. King & M. Waldén, eds, 'ZB 2003: Formal Specification and Development in Z and B / Third International Conference of B and Z Users', Vol. 2651 of *Lecture Notes in Computer Science*, Springer-Verlag Heidelberg, pp. 148–167.

Deutsch, M., Henson, M. C. & Reeves, S. (2002), Six theories of operation refinement for partial relation semantics, Technical Report CSM-363, Department of Computer Science Department, University of Essex.

Deutsch, M., Henson, M. C. & Reeves, S. (2003), Operation refinement and monotonicity in the schema calculus, *in* D. Bert, J. P. Bowen, S. King & M. Walden, eds, 'ZB 2003: Formal Specification and Development in Z and B', Vol. 2651 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 103–126.

Dijkstra, E. W. (1976), *A Discipline of Programming*, Prentice Hall.

Harel, D. (1987), 'Statecharts: A visual formalism for complex systems', *Science of Computing* pp. 231–274.

Henson, M. C., Deutsch, M. & Kajtazi, B. (2004), The specification logic νZ, Technical Report CSM-421, Department of Computer Science, University of Essex.

Henson, M. C. & Reeves, S. (2000), 'Investigating Z', *Journal of Logic and Computation* **10**(1), 1–30.

Henson, M. C. & Reeves, S. (2003), 'A logic for schema-based program development', *Formal Aspects of Computing Journal* **15**(1), 48–83.

Philipps, J. & Scholz, P. (1997*a*), Compositional specification of embedded systems with statecharts, *in* M. Bidoit & M. Dauchet, eds, 'TAPSOFT '97: Theory and Practice of Software Development', number 1214 *in* 'LNCS', Springer-Verlag, pp. 637–651.

Philipps, J. & Scholz, P. (1997*b*), Formal verification of statecharts with instantaneous chain reaction, *in* E. Brinksma, ed., 'Tools and Algorithms for the Construction and Analysis of Systems', Vol. 1217 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 224–238.

Reeve, G. (2005), μCharts-Investigating Refinement (To appear), PhD thesis, Department of Computer Science, University of Waikato.

Reeve, G. & Reeves, S. (2000), μ-Charts and Z: Hows, whys and wherefores, *in* W. Grieskamp, T. Santen & B. Stoddart, eds, 'Integrated Formal Methods 2000: Proceedings of the 2nd. International Workshop on Integrated Formal Methods', LNCS 1945, Springer-Verlag, pp. 255–276.

Scholz, P. (1998), A refinement calculus for statecharts, *in* E. Estesiano, ed., 'Fundamental approaches to software engineering: First International Conference, FASE'98', Vol. 1382 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 285–301.

Spivey, J. M. (1989), *The Z notation: A reference manual*, Prentice Hall.

Woodcock, J. & Davies, J. (1996), *Using Z: Specification, Refinement and Proof*, Prentice Hall.

Z-lambda project (2005).
URL: *www.cs.waikato.ac.nz/Research/fm*

Supporting Software Reuse by the Individual Programmer

Min-Sheng (Peter) Hsieh†, Ewan Tempero‡
Department of Computer Science
University of Auckland
Auckland, New Zealand
†mhsi005@ec.auckland.ac.nz
‡e.tempero@auckland.ac.nz

Abstract

Despite its long history and its benefits, software reuse has yet to become a common practise among software programmers. While there is much ongoing research, it focuses on large-scale organisation-level techniques and methodologies. There is very little research that considers reuse at the personal level as an important factor. The lack of focus and tool support has limited the potential for developers to reuse their past efforts. This paper introduces ICRT (Individual Code Reuse Tool), which provides support for an individual to efficiently reuse code fragments written in the past. ICRT uses the CBR methodology to manage the code fragments, and is integrated with the Eclipse IDE.

Keywords: Code Reuse, Case-Based Reasoning, Tool Support

1 Introduction

Consider the following scenario: Chris has been given the task of writing some code that creates a user interface that requires a somewhat complex layout of its component parts. As she begins writing the code, she realises that what she is doing is similar to what she did for a previous project. She quickly finds the code she wrote for that project, and confirms that large parts of it are relevant to her current project. By judicious cutting, pasting, and adaption, she is able to produce the code she needs much more quickly than if she had continued developing it from scratch.

Any programmers who have written code will be familiar with the above scenario. It is situations like this that provide an opportunity for improved productivity by avoiding writing that same code, if only the programmers could quickly access their previous efforts to solve the problem at hand. Such opportunities have been recognised since the beginnings of software engineering, and there has been much research in *software reuse*, that is, in how to leverage such opportunities. Much of this research has focused on how organisations can gain the benefits of reuse. There have, however, been comparatively few efforts that support the individual programmer reusing her own past efforts.

In this paper, we present the results of our investigation into providing reuse support for the individual programmer. Specifically, we discuss ICRT, an Individual Code Reuse Tool. This tool stores *code fragments* developed by the programmer's past efforts and uses the case-base reasoning (CBR) methodology for retrieval of the most relevant piece to the current effort. ICRT is integrated into the Eclipse IDE with particular attention being paid to usability so as to minimise the cost of its use to the programmer.

The paper is organised as follows. In the next section, we introduce the important concepts, in particular what is needed to develop a CBR-based system. We also discuss other related research. In section 3, we present the motivation for this work in more detail and discuss how that affects the requirements for ICRT. Section 4 presents the main concepts we use to develop a CBR-based system targeted at reusing source code. We then give an overview of ICRT in section 5. Section 6 presents an evaluation of ICRT, and finally, we discuss future work and present our conclusions.

2 Background and Related Work

2.1 Software Reuse

The idea of developing software by re-using existing software has been around since the dawn of software engineering as a discipline (McIlroy 1969). Since that time, much research has been done to turn this idea into reality, of which we can only touch on here (see surveys such as (Mili, Mili & Mili 1995, Kim & Stohr 1998) for more detail). For the most part this research has focused on planned or *systematic reuse*, that is, how organisations can, by using explicit processes and standards, get the most benefit from reuse. Early efforts in this regard examined the design, development, and organisational use of repositories of reusable assets (see (McClure 1997) for example). Later efforts considered domain engineering and domain analysis as promising avenues (see (Tracz, Coglianese & Young 1993) for example), which led to software product lines (Clements & Northrop 2001).

There has also been much work in tool support for reuse. The early work concentrated on repository support. Later work examined other aspects of supporting reuse, such as the development of reusable assets and reducing the cost of understanding them (for example, (Biddle & Tempero 1998)). Other work has provided support for non-code assets, for example design patterns (Mapelsden, Hosking & Grundy 2002).

The tools developed in this kind of research, while used by an individual, have been intended to support reuse at the organisational level. In fact, most Software Reuse researchers have ruled out supporting the kind of reuse exemplified by our scenario, describing it as *software salvaging* and in doing so implying it is not worth supporting (Tracz 1995). However, as we (and many others) have noted, reuse often occurs at the individual level, where an individual leverages her own past work rather than use an enterprise level repository.

For this kind of reuse, there appears to have been little work done in developing tools to support it. One exception is work by Norton on "Reuse of personal software assets"(Norton 2003). Norton observed that an individual builds up her own personal collection of useful assets during her career, and that it was feasible to provide support to help manage that collection. He identified seven features that a tool providing this support should have: privacy, Internet accessibility, customisable meta-data, flexible browsing, support for a variety of asset types, support for relationships between assets, and a natural language query facility. We compare Norton's work with ours in more detail in section 6.

2.2 Case-based Reasoning (CBR)

CBR involves reasoning from prior experience: retaining a memory of previous problems and their solutions, and solving new problems by reference to that knowledge. Generally, a CBR system will be presented with a problem, either by a user or by a program or system. The system then searches its memory of past cases (called the "case-base") and attempts to find a case that has the same problem specification as the case under analysis. If the reasoner cannot find an identical case in its case-base, it will attempt to find a case or multiple cases that most closely match the current problem (Pal & Shiu 2004).

Pal and Shiu discuss many advantages in adopting CBR (Pal & Shiu 2004). In particular, a CBR system can still function even if the underlying theory of domain knowledge has not been quantified or understood entirely. Availability is another key advantage of having CBR systems. Whereas artificial intelligence techniques require full knowledge of the domain to be available, CBR can be usefully used even then there is only a few cases in its case-base.

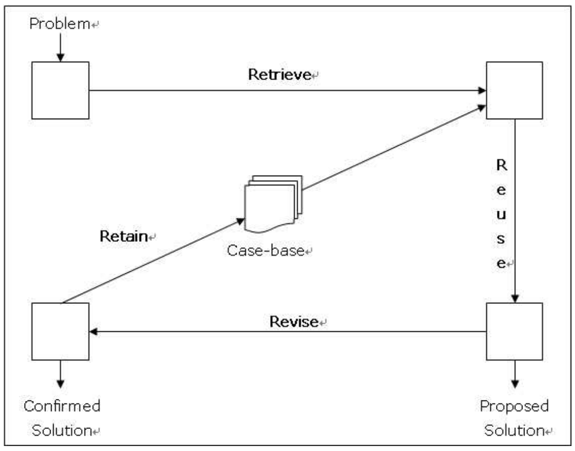


Figure 1: The CBR cycle

The process involved in retrieving past experience can be generalised into four consecutive steps, known as the CBR Cycle, as shown in figure 1 (Watson 1997).

In order for cases stored in a case-base to be retrieved during queries, each case must have a standard representation and it must be indexed properly.

Case representation is the first step in implementing a CBR system. It is also the most important step because the representation will reflect the knowledge stored in each case. In many practical CBR applications, cases are usually represented as two unstructured sets of attribute–value pairs that represent the problem and solution features (Gebhardt, Vob, Grather & Schmidt-Beltz 1997). One of the advantages of CBR is that it allows flexibility in how an attribute can be represented. There is a range of possible choices from simple boolean, numeric and text data, to binary files, time-dependent data and relationships between data.

It is not enough to just represent the cases, a structure representation for the case-base is required as well. This structure will greatly influence how the index is constructed. There are two common structures used: flat or hierarchical. A flat structure has the property that the indexes are chosen to represent the important aspects of the case and retrieval involves comparing the query case's attributes to the attributes of each case in the case-base. A hierarchical structure, on the other hand, stores the cases by grouping them into appropriate categories to reduce the number of cases that have to be searched during a query (Pal & Shiu 2004).

Case indexing is the second step of implementing CBR. In this step cases are indexed for future retrieval and comparison. The choice of indexes is important to enable retrieval of the right case at the right time because the indexes will determine in which context a case will be retrieved in the future. Therefore it is critical that the indexes reflect the important features of a case and the attributes that influence the outcome of the case, and describe the situations in which a case is expected to be retrieved in the future (Pal & Shiu 2004). Although there are attempts at making the process automatic, generally indexes are assigned by domain exports.

Case retrieval is the process of finding cases that are closest to the current case. In order to carry out effective case retrieval, there must be selection criteria that determine how a case is judged to be appropriate for retrieval and a mechanism to control how the case-base is searched (Pal & Shiu 2004).

Nearest neighbour retrieval (NNR) is a common selection technique applied in CBR systems. The algorithm will retrieve a case when the weighted sum of its features that match the current query is greater than other cases in the case-base (Pal & Shiu 2004).

Once a case has been chosen, it will be modified or *adapted* to fit into the current problem. If the underlying domain of the cases is well understood, then sometimes this step can be automated, but often case adaptation is performed manually.

3 Requirements

The main motivation for the development of ICRT is to support the reuse of code that a programmer has written in the past. To understand that this means in terms of the specific requirements for the design of ICRT, we need to discuss in more detail what it means to provide this kind of support, and what some of the consequences are.

As we mentioned in the introduction, the situation ICRT is meant for is when a programmer thinks she is about to write code that might similar to code she has written in the past. The first point to note is that we are only interested in source code, and not other artifacts that could be considered useful for reuse. The next point to note is that we are not just interested in reuse of "whole" pieces of code, that is, semantically complete code such as classes, modules or components, but also code fragments that may not even be syntactically correct. This impacts the choice of technology for representing and storing the artifacts we wish to reuse.

Reusing code is only useful if the total cost of doing so is less than the cost of creating the code from scratch. So what is the cost of reuse? In the situation above, if the programmer knows the exact location of code that will do exactly what is needed, then reusing it will not cost much, although there is still the cost of navigating to the location and of integrating the code into the current context.

If there is uncertainty about the location or relevance of the existing code, then reusing it becomes more expensive. It does no good if the programmer has written exactly the code she needs some time in the past, but it takes her two hours to find it (she eventually discovers it has been moved to an off-line archive) when she could have re-written it in an hour. It also does no good if the programmer takes only 15 minutes to find the code she remembered, only to realise it's of no use and have to start from scratch anyway. The greater the uncertainty of location or the relevance of existing code, the less likely the programmer will even attempt to try to reuse code. This leads to the primary requirement for ICRT, to quickly identify the most relevant existing piece of code to the problem at hand.

```
JTable table = new JTable( 15, 3 ) {
    public Component prepareRenderer(TableCellRenderer renderer,
                                     int row, int column)
    {
        Component c = super.prepareRenderer( renderer, row, column);
        // We want renderer component to be transparent so background
        // image is visible
        if( c instanceof JComponent )
            ((JComponent)c).setOpaque(false);
        return c;
    }
};

// Use our version of JScrollPane
MyScrollPane sp = new MyScrollPane( table );

// Set the background image
ImageIcon image = new ImageIcon( "codeguruwm.gif" );
sp.setBackgroundImage( image );
```

Figure 2: A code fragment to be stored in ICRT

Another cost of reuse takes place after a relevant code fragment has been identified, and that is the cost to adapt and integrate it into the current context. This cost can be reduced by initially writing the code code fragment to be easy to reuse, however this makes the initial creation of the fragment to be more expensive. Typically, there is a trade-off between the development cost and the cost of adaption and integration. If the code is reused several times, then its up-front cost can be amortised over its lifetime. However, it is often difficult to predict whether code is going to be reused, and so there is a risk that the cost of making the code easier to reused will never be recovered because the code is never reused. A way to reduce this cost is to spread the cost of making the code easier to reuse over its lifetime. Every time code is reused, it is improved in a way that will (hopefully) make it easier to reuse in the future, and the improved version is kept for future reuse requirements. This leads to the requirement for ICRT of providing support for retaining these improved versions.

We are proposing the use of a new tool to support code reuse. This is in itself a cost. If the tool takes a long time to learn or takes a significant amount of effort to use, then it is unlikely to be actually used. This means that good *usability* is an important consideration for ICRT. Usability is now recognised as an important factor in the lack of use of CASE tools (Iivari 1996). In particular, it must be easy to store code in the tool when it is first created, easy to search for relevant code, easy to integrate relevant code, and easy to retain modified versions.

Finally, it must be kept in mind that the tool we're proposing is intended to support reuse at the level of the individual. This has important implications in what aspects of the tool's design are important. One is the usability issue mentioned above. With systematic reuse, a company's reuse policy may get away with dictating the use of a tool that no-one enjoys using, since there will be external pressures (such as keeping one's job) to ensure the policy is followed. However if the individual is deciding whether or not to use a tool, then that individual's experience with the tool will affect the decision.

Another consequence of providing individual support is that there is no need to enforce any standard. In fact, it is likely that the tool would be more acceptable if the user could tailor its behaviour to suit her particular way of working. Different people classify and remember things in different ways, so we have concluded that, rather than provide a rigid indexing structure, we should instead give some control of the indexing to the user.

4 CBR in ICRT

In general, as others have noted, there are similarities between the CBR and reuse approaches (Tautz & Althoff 1997). However the CBR methodology seems a particularly good match to ICRT's requirements. The artifacts we wish to reuse, code fragments, have little useful structure on which to base a useful domain model. We expect ICRT to be useful right from the start when there are only a few fragments available. We want to be able to provide support for retaining new versions of code created when they are adapted for reuse, something that is explicit in the CBR cycle.

The process of applying CBR as mentioned in section 2.2 can be simplified into four steps: case representation, case indexing, case retrieval and case adaptation.

Starting with case representation, cases in ICRT consist of attribute-value pairs, where the value is the code fragment represented as a string, and the attribute describes the functionality as discussed below. We chose to represent the code as unstructured strings, instead of, for example, representing the semantics in some way, because unstructured strings don't have to be even syntactically correct and yet can still be useful for our purposes. Furthermore, from a usability perspective, operations such as copy and paste are familiar to our user population, and these operations are based on unstructured strings. It also provides an opportunity for a simpler code fragment retrieval interface, as well as more efficient case indexing. The cases are stored in a flat structure.

In existing CBR systems, the size of the index is the same for each case. This is because each case is expected to have a fixed set of features. The index for a case is then based on the values of each feature for that case. However, in the case of indexing a code fragment there is no standard or obvious feature set that can be used to describe each fragment. Therefore we use a indexing mechanism such that fragments can be assigned an arbitrary (but user specified) number of features.

We call each feature/property pair a *functionality card*. Currently the cards used in ICRT have a simple structure: [*Language — Feature — Property — Description*]. For instance, a code fragment that might be described as "Reading a file from a given location" can be represented as: [*Java—I/O—Read—From a given location*]. In this case, the feature on the card also refers to the corresponding Java IO package. This structure originated from Java Almanac (Chen 2002) and it has been modified to fit into the purpose of the card design. The purpose of a functionality card is to document functionality that is meaningful to the user. This means that the card deck created by the user will be unique to that user. It is possible to have repeated cards between programmers but they may mean different things depending on the users' preferences.

The card approach has the benefit that it requires less computation in case retrieval compared to text-based description because there is no need to perform text processing to pick up the key words in both the input query and within the cases. It also simplifies both the indexing and retrieval process, which we believe improves usability. Rather than learn and use a complex syntax or search process, a user just picks out the cards the best represent the functionality she is interested in (for retrieval) or best represents the code fragment (for indexing).

For case retrieval, ICRT uses nearest neighbour retrieval. By default, cards have equal weights, but the weights can be changed. Case adaptation in ICRT is currently performed manually.

To see how the cards work, consider the code fragment presented in figure 2. The language is *JAVA*, and generally the code deals with user interface elements. However some of it (`JTable`) is specific to the `swing` package, whereas the last two statements are more general. When indexing, the user might choose to separate these into separate features, such as *SWING* and *GUI*. The properties provide more specific information about the code fragment with respect to the identified features, and so the cards that assigned to this fragment could as follows:

- [*JAVA — SWING — Table — Java Swing Table related i.e., JTable*]
- [*JAVA — GUI — Background — Embed background in GUI components*]
- [*JAVA — GUI — Image — GUI component relating to Image*]

This choice of cards is, in keeping with the comments in section 3, completely up to the user. Another user may have instead decided that the "image" aspect of the code was not worth recording, and so left that card out, or that the transparency of the cells was relevant, and so included a card representing that information. Even the form of the cards is up to the user. She might decide that managing the background of a user interface was sufficiently important that it become a feature, rather than a property.

5 Individual Code Reuse Tool (ICRT)

In this section, we demonstrate ICRT. As previously mentioned, ICRT is a plug-in for the Eclipse IDE, and consequently uses parts of Eclipse, such as the SWT. Code fragments are stored in HSQLDB (IBM 2005). Although HSQLDB is not as feature rich as other database management systems, it has the benefits that it can operate without a separate database server having to be installed. ICRT uses Hibernate provide the mapping between the relational data model used in HSQLDB and the representation used in ICRT (JBoss 2005).

Figure 3 shows ICRT as a user would see it. It is presented as an Eclipse **View** shown in the figure as one of a set of tabbed panes in the frame at the bottom of the interface. ICRT itself has a number of views, for code retrieval and indexing, for card management, and for improving code.

5.1 Procedures involved in Code Retrieval

There are four steps involved in retrieving a code fragment:

1. Select appropriate functionality cards
2. Assign weights (or importance) on the selected cards, if necessary
3. Select a code fragment from the fragment list and the user can preview the fragment's content in the code preview field
4. Press the copy button to copy the fragment to clipboard and the user can decide where to paste the fragment

The fragment search is performed automatically whenever the user adds/removes a card to/from the selected card list. All fragments that have the chosen card will be listed.

5.2 Sample Scenario

To see how ICRT would be used, consider the following scenario. Chris is assigned with the task of placing her company logo into a Java accounting application, which consists mainly of Java swing tables. Chris vaguely remembers that she has a code fragment stored in ICRT that performs similar task and she believes that the same fragment can be used here.

She starts off ICRT in her Eclipse IDE and begins her code retrieval process. She will need to pick the cards that are directly associated with the task, that is. embedding background image in a JTable. She first picks the Background card in the GUI category first as the diagram shown in Figure 4.

After selecting the Background card, 5 code fragments are shown in the result list. But with only the background card selected, Chris finds it difficult to distinguish fragments relating to background colour and background image. Therefore she selects the Image card in the same category, which in turn emphasises that she is more interested in code fragments relating to background image. By doing so, fragments relating to both Background and Image will now have 100% similarity rate whereas those that relate to Background and Colour will only have 50% similarity. Note that a new code fragment (one that matches Image but not Background) has also been added, but also only with 50% similarity.

From the descriptions shown in Figure 5 Chris cannot tell which fragment is for Java Swing tables, so she selects the Table card in the Swing category as shown in Figure 6. Although adding the Table card results more code fragments appearing in the fragment list (adding those for Table), Chris only needs to investigate the fragments that have the highest similarity percentage as they are more likely to be the fragment she is interested in using. In this case, Chris sees two possible solutions to her current problem, she could choose either the "scrollable" or the "non-scrollable" background image solution.

This example illustrates how code retrieval is performed in ICRT. As one can see that the user does not need to type out her query, instead ICRT allows the user to could simply choose the relevant cards. The additional benefit of using cards is that there is no worry of possible typos, which could cause no fragments found.

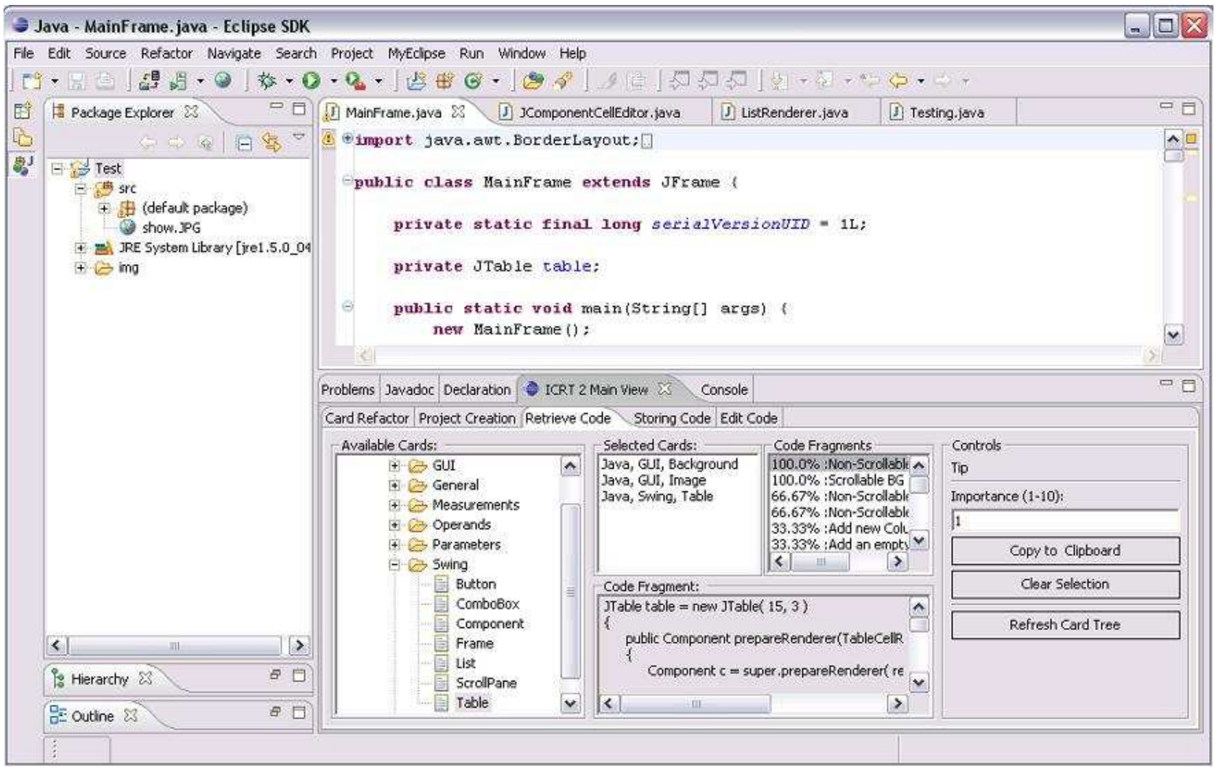
5.3 Other features supported in ICRT

Apart from the main code retrieval functionality, ICRT also provides other features, such as code storing and new functionality card creation. These features are essential to how a user may retrieve her code fragment from ICRT.

5.3.1 Storing Code In ICRT

Code storage and retrieval are the two most important functionalities that ICRT addresses. We have outlined the procedures involved in code retrieval in the previous section, therefore in this section we will discuss the process involved in storing new code fragments into ICRT. A screen shot of the interface is shown in Figure 7.

The process involved in storing a new code fragment consists of six steps:



```
import java.awt.BorderLayout;

public class MainFrame extends JFrame {

    private static final long serialVersionUID = 1L;

    private JTable table;

    public static void main(String[] args) {
        new MainFrame();
```

Selected Cards:

```
Java, GUI, Background
Java, GUI, Image
Java, Swing, Table
```

Code Fragments

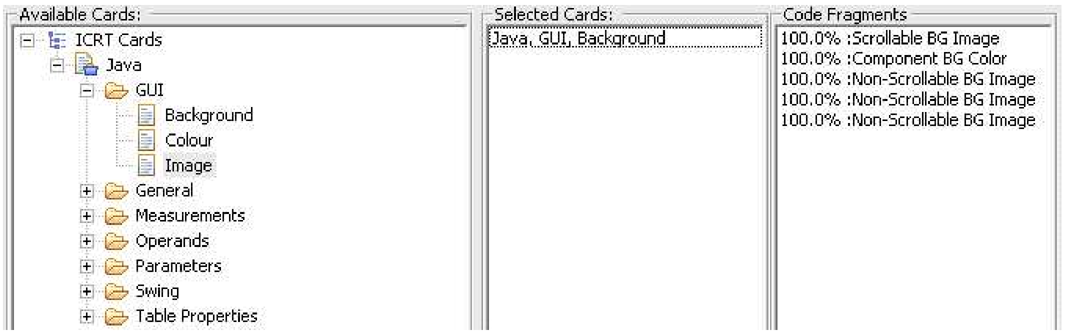
```
100.0% :Non-Scrollabl
100.0% :Scrollable BG
66.67% :Non-Scrollabl
66.67% :Non-Scrollabl
33.33% :Add new Colu
33.33% :Add an empty
```

Code Fragment:

```
JTable table = new JTable( 15, 3 )
{
    public Component prepareRenderer(TableCellR
    {
        Component c = super.prepareRenderer( re
```

Importance (1-10): 1

Figure 3: ICRT as an Eclipse Plug-in



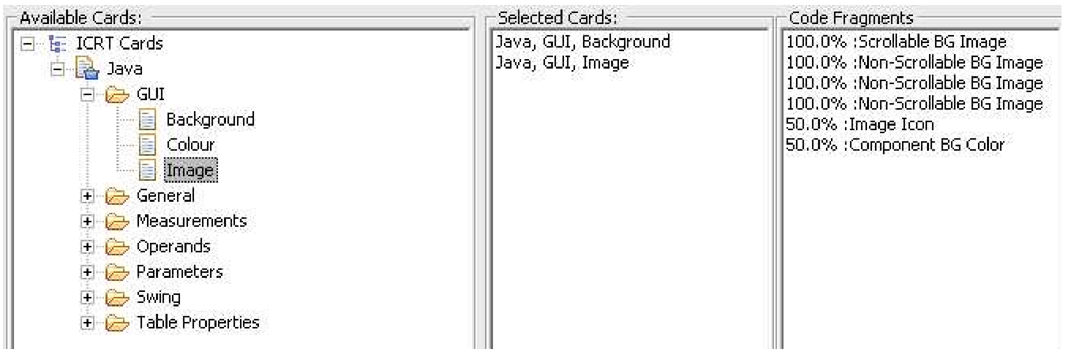
Selected Cards:

```
Java, GUI, Background
```

Code Fragments

```
100.0% :Scrollable BG Image
100.0% :Component BG Color
100.0% :Non-Scrollable BG Image
100.0% :Non-Scrollable BG Image
100.0% :Non-Scrollable BG Image
```

Figure 4: First card selected in code retrieval



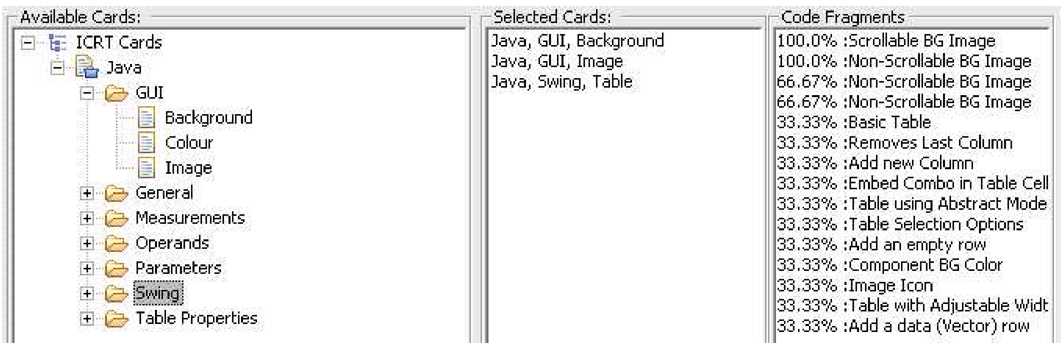
Selected Cards:

```
Java, GUI, Background
Java, GUI, Image
```

Code Fragments

```
100.0% :Scrollable BG Image
100.0% :Non-Scrollable BG Image
100.0% :Non-Scrollable BG Image
100.0% :Non-Scrollable BG Image
50.0% :Image Icon
50.0% :Component BG Color
```

Figure 5: Second card selected in code retrieval



Available Cards:
- ICRT Cards
  - Java
    - GUI
      - Background
      - Colour
      - Image
    - General
    - Measurements
    - Operands
    - Parameters
    - Swing
    - Table Properties

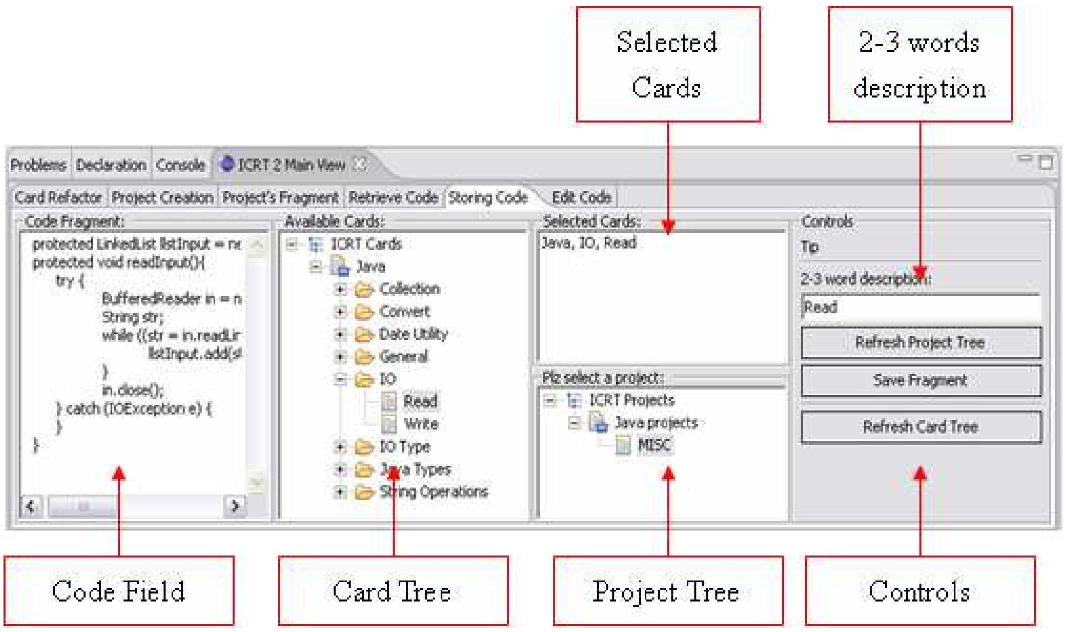
Selected Cards:

Java, GUI, Background
Java, GUI, Image
Java, Swing, Table

Code Fragments

```
100.0% :Scrollable BG Image
100.0% :Non-Scrollable BG Image
66.67% :Non-Scrollable BG Image
66.67% :Non-Scrollable BG Image
33.33% :Basic Table
33.33% :Removes Last Column
33.33% :Add new Column
33.33% :Embed Combo in Table Cell
33.33% :Table using Abstract Mode
33.33% :Table Selection Options
33.33% :Add an empty row
33.33% :Component BG Color
33.33% :Image Icon
33.33% :Table with Adjustable Widt
33.33% :Add a data (Vector) row
```

Figure 6: Third card selected in code retrieval



```
protected LinkedList listInput = ne
protected void readInput(){
      try {
            BufferedReader in = n
            String str;
            while ((str = in.readLir
                  listInput.add(st
            }
            in.close();
      } catch (IOException e) {
      }
}
```

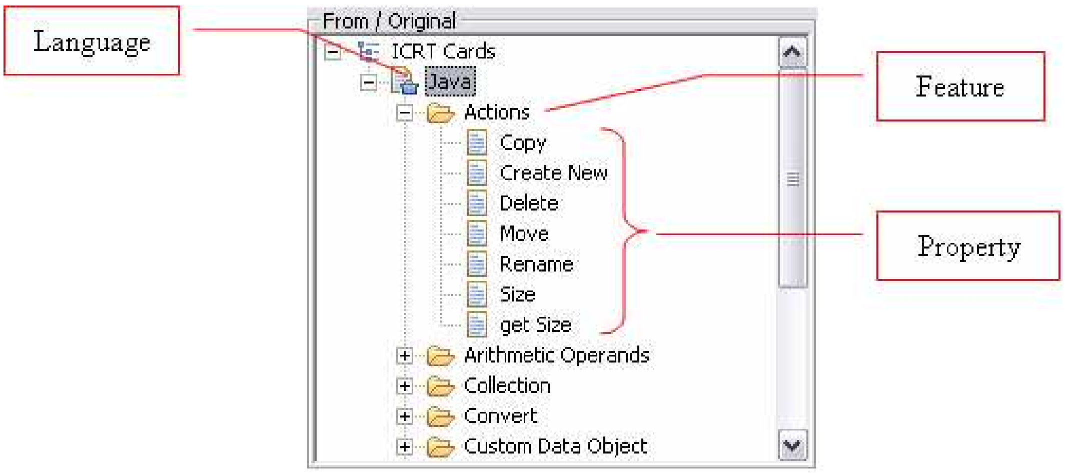
Available Cards:
- ICRT Cards
  - Java
    - Collection
    - Convert
    - Date Utility
    - General
    - IO
      - Read
      - Write
    - IO Type
    - Java Types
    - String Operations

Selected Cards: Java, IO, Read

2-3 word description: Read

Plz select a project:
- ICRT Projects
  - Java projects
    - MISC

Figure 7: Store new code fragment



- ICRT Cards
  - Java
    - Actions
      - Copy
      - Create New
      - Delete
      - Move
      - Rename
      - Size
      - get Size
    - Arithmetic Operands
    - Collection
    - Convert
    - Custom Data Object

Figure 8: Functionality Card Tree

1. Copy the code fragment into the code field
2. Make necessary changes to the code, e.g. generalisation
3. Select the cards representing the functionalities of the code from the card tree
4. Choose a project
5. Write two to three words describing the fragment
6. Press the "Save Fragment" button

In step five, we allow a developer to write two to three words describing on the code fragment so that when this fragment, along with other relevant fragments, is retrieved, she will be able to identify the fragment that she has the most interest in quickly.

5.3.2 Card Management In ICRT

The Functionality card design plays a big role in ICRT. It enables a developer to retrieve code quickly and to store new code fragments easily by selecting the relevant cards. One of the costs of the flexibility the user has in the choice of cards is that wrong choices may be made. For example, a user may initially decide that "background" is useful as a feature, but later realise that it is much too narrow and so would be better off as a property. This suggests there needs to be support for the management of cards.

As shown in previous figures the cards are presented in a tree structure. As shown in Figure 8 the first level of tree nodes represents the Language category, followed by Feature and the leaves of the tree represent the Properties. Since a card is made up of Language, Feature and Property, this structure allows a developer to quickly identify the card she is looking for. When the developer leaves her mouse on top of a selected node she will be able to see a tool tip showing, which contains information about selected node.

There is also the problem that it is very easy for a developer to fall into the trap of creating a new card for each new code fragment stored. Therefore we designed the card refactor interface to allow a user to "refactor" her card set to reduce redundancies. Apart from the general create, edit and delete functionalities, we provide three additional features that will enable the developer to refactor her cards more easily. They are:

- Copy functionality allows the developer to copy her cards to another language category, which saves her time and effort in creating them individually.
- Move functionality allows the developer to move her cards to another location. Once this operation is completed, the original cards will be removed.
- Replace functionality allows the developer to replace a selected card with other cards. This will allow the user to "refactor" her cards for example, replacing a specific card with other more general cards.

There are other aspects of ICRT. Of particular interest is the decision to clear the selected cards once the user presses the copy button. This behaviour was added after observing incorrect use of the tool by users, as discussed in the next section.

6 Evaluation and Discussion

We performed a formal evaluation of ICRT. The participants were given a task to do involving simple string-based manipulation of dates. While ICRT has been designed with the idea that users add and organise code fragments themselves, time constraints meant that we could not simulate that in our study. Instead, the participants

2 code fragments from Java SQL package (e.g. SQL date)
3 code fragments on Java String operations (e.g. Split string)
2 code fragments on Java Swing (e.g. Frame)
3 code fragments on General Java operation (e.g. main, and for loop)
3 code fragments on Debugging (e.g. print list)
6 code fragments on Date Calculations (e.g. get particular day of week)
7 code fragments on Converting Object to another (e.g. String to Date)
16 code fragments on Java I/O operations (e.g. read in and write out)

Table 1: Code fragments for evaluation

One 2nd year undergraduate student
One 3rd year undergraduate student
One graduate student with a Bachelor of Engineering (4 year degree)
Two people from industry (both graduated more than a year previously)

Table 2: Evaluation participants

where given ICRT already populated with 42 fragments (Table 1), some of which were relevant to the task. There were 5 participants with a variety of backgrounds (Table 2). At the end of the study, the participants filled in a questionnaire. We summarise the results of the most relevant questions below.

Q 14 *Do you think you can program faster without the support of ICRT? Yes/No*

All the participants selected "No".

Q 15 *How easy/hard is it for you to operate ICRT? Rank from 1-5 (1 means very easy, 5 means very difficult) and what improvements would you like to see in ICRT?*

The responses were generally positive, although several commented on the difficulty relating to not having organised the code fragments themselves. This confirms our belief that users prefer to do their own organisation. The complete set of responses is shown in Table 3.

Q 16 *What do you think about the card-based searching? Rank from 1-5 (1 means very easy, 5 means very difficult)? Could you write a few words explaining your answer?*

Again we saw comments that are related to the participants not having organised the code fragments themselves. They also raised questions about how well ICRT would work with many cards. This is something we would like to address in future studies. The complete set of responses is shown in Table 4.

One aspect of ICRT that was noticed while observing participants in the study was that they frequently left cards they had used for previous queries selected when making new queries. This meant that the effectiveness of the search was reduced as they were mixing different functionalities. As a consequence, we have changed ICRT to clear the selected cards whenever a fragment is copied in the buffer.

Participant ID	Rank	Comments made by participants
1	2	It is something that would get better with time, i.e., contains more code. I like it.
2	2	Enable the user to clear a particular code fragment that has been used
3	3	Handy if I have my own cards build up
4	1	I am not sure if ICRT dynamically updates the database as code is entered. It is good functionality to have.
5	2	It would have been even easier had I organised the cards myself

Table 3: Q 15 on ICRT usability

Participant ID	Rank	Comments made by participants
1	3	Initially spend some time understanding what each card was for. It's due to experience with the tool
2	3	It is easy to fi nd fragments using the cards provided. However in the case where large numbers of cards are kept within the tree, it is best to have functionalities such as key word search for the cards.
3	2	Because it increases my coding effi ciency (i.e., less time involved in searching for code fragment)
4	1	Convenient. But can get diffi cult to search if there are too many cards
5	2	(Refer to 15) Having text-based searches in addition to cards might be quicker, especially if the persona using ICRT did not organise the cards

Table 4: Q 16 on ICRT Card design

ICRT differs from Norton's PARSE system (Norton 2003) in several ways. In particular, ICRT only provides support for reuse of code fragments, whereas, not only does Norton provide support for multiple asset types, he suggests that such support is a key feature of any personal asset reuse support solution. We disagree. We believe it is better to provide very good support for an asset type that is commonly reuse, namely code fragments. In order to provide very good support, we also focus much more on usability. This focus has affected our decision as to what assets are stored, and how users interact with them. It also led to the decision to integrate ICRT with an existing IDE. ICRT has been designed with a view to minimise the amount of time users spend searching, adding, and retrieving assets. To this end, ICRT uses CBR to manage the reusability assets in the hope that this will yield high quality search results.

Norton briefly touched on other currently available software products that also store code fragments. Out of 19 tools he lists, Code Keeper (ICY 2005) is most similar to ICRT in terms of functionality. This tool allows a user to browse, search and store code fragments. There is one major difference between ICRT and this tool is that ICRT is integrated with an IDE, whereas Code Keeper is standalone. It is also unclear how effective its search facilities are.

7 Conclusions and Future Work

We have introduced ICRT, a plug-in to the Eclipse IDE intended to support the individual programmer reusing code she has written in the past. Specifi cally, ICRT provides support for a task that many programmers currently do anyway, namely fi nding and reusing code they have written in the past.

The support provided by ICRT uses the CBR methodology. To do so we represent cases as fragments of code together with a set of "cards", which are feature/property pairs. The cards provide a simple mechanism for indexing the fragments and the searching through the case base. Matches are done using the nearest neighbour algorithm. Usability is an important requirement of ICRT. The decisions to restrict to supporting reuse of just code fragments, the cards mechanism, and the integration with an IDE, are all intended to meet this requirement. We have carried out an evaluation of ICRT using a small but diverse set of users. While the sample size was too small to draw statistically signifi cant conclusions, the feedback was positive and provides useful direction for future work.

There is also an issue that we anticipate will require further work. Currently, ICRT requires the user to choose appropriate cards whenever a new fragment is added. This takes some time, and poor choice can affect the effectiveness of later searches. This will become more of an issue as the number of cards increases. We would like to explore either automated, or semi-automated card assignment. Finally, the general question of whether or not ICRT improves productivity is yet to be empirically answered. Given that ICRT automates something programmers already do, there is good reason to believe it does and we have some evidence to this effect, however more work has to be done.

Acknowledgements

The authors would like to thank the participants in the experiment for their time, and also the useful comments from the referees.

References

Biddle, R. L. & Tempero, E. D. (1998), Towards tool support for reuse, *in* 'SEEP '98: Proceedings of the 1998 International Conference on Software Engineering: Education & Practice', IEEE Computer Society, p. 126.

Chen, P. (2002), *Java(TM) Developers Almanac 1.4, Volume 1: Examples and Quick Reference*, Addison-Wesley Professional.

Clements, P. & Northrop, L. M. (2001), *Software Product Lines: Practices and Patterns*, Addison Wesley.

Gebhardt, F., Vob, A., Grather, W. & Schmidt-Beltz, B. (1997), *Reasoning with Complex Cases*, Kluwer Academic, Norwell, MA.

IBM (2005), 'Eclipse and HSQLDB: Embedding a relational database server into eclipse, part 1'. `http://www-106.ibm.com/developerworks/opensource/library/os-echsql/?ca=%lnxw09HSQLDB`.

ICY (2005), 'Code keeper 1.0'. `http://www.icynorth.com/codekeeper/`.

Iivari, J. (1996), 'Why are CASE tools not used?', *Communications of the ACM* **39**(10), 94–103.

JBoss (2005), 'Introducing hibernate'. `http://www.hibernate.org/4.html`.

Kim, Y. & Stohr, E. A. (1998), 'Software reuse: Survey and research directions', *Journal of Management Information Systems* **14**(4), 113–147.

Mapelsden, D., Hosking, J. & Grundy, J. (2002), Design pattern modelling and instantiation using dpml, *in* 'CRPITS '02: Proceedings of the Fortieth International Confernece on Tools Pacifi c', Australian Computer Society, Inc., pp. 3–11.

McClure, C. (1997), *Software Reuse Techniques*, Prentice Hall.

McIlroy, M. D. (1969), Mass produced software components, *in* P. Naur & B. Randell, eds, 'Proceedings of NATO Software Engineering Conference', Vol. 1, NATO Science Committee, pp. 138–150. Presented at the NATO conference on software engineering, Garmisch, Germany, 7-11 October, 1968.

Mili, H., Mili, F. & Mili, A. (1995), 'Reusing software: Issues and research directions', *IEEE Transactions on Software Engineering* **21**(6), 528–561.

Norton, R. J. (2003), Reuse of personal software assets: Theories, practices and tools, Master's thesis, The Florida State University.

Pal, S. K. & Shiu, S. C. K. (2004), *Foundations of Soft Case-Based Reasoning*, John Wiley & Sons.

Tautz, C. & Althoff, K.-D. (1997), Using case-based reasoning for reusing software knowledge, *in* 'Second International Conference on Case-Based Reasoning Research and Development', pp. 156–165.

Tracz, W. (1995), *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*, Addison-Wesley.

Tracz, W., Coglianese, L. & Young, P. (1993), 'A domain-specifi c software architecture engineering process outline', *SIGSOFT Softw. Eng. Notes* **18**(2), 40–49.

Watson, I. (1997), *Applying Case-based Reasoning Techniques for Enterprise Systems*, Calif.: Morgan Kaufmann.

Identifying Refactoring Opportunities by Identifying Dependency Cycles

Hayden Melton, Ewan Tempero
Department of Computer Science
University of Auckland
Auckland, New Zealand
{hayden|ewan}@cs.auckland.ac.nz

Abstract

The purpose of refactoring is to improve the quality of a software system by changing its internal design so that it is easier to understand or modify, or less prone to errors and so on. One challenge in performing a refactoring is quickly determining where to apply it. We present a tool (Jepends) that analyses the source code of a system in order to identify classes as possible refactoring candidates. Our tool identifies dependency cycles among classes because long cycles are detrimental to understanding, testing and reuse. We demonstrate our tool on a widely-downloaded, open-source, medium-sized Java program and show how cycles can be eliminated through a simple refactoring.

1 Introduction

Refactoring is defined as "the process of changing a software system in such a way that does not alter the external behaviour of the code yet improves its internal structure" (Fowler 1999). Refactoring is most appropriate for software systems whose existing (internal) design is hard to understand, hard to modify and prone to errors and so on. By refactoring such a software system we alter its design to make it easier to understand, modify and less prone to errors. As such, refactoring is regarded as an important technique for improving software quality during a system's maintenance phase.

There are several challenges in performing a refactoring. One is to identify characteristics of a design that make it hard to understand, modify or test etc. Fowler produces a list of these characteristics which he refers to as 'bad smells in code' or simply smells. Examples of smells include large classes, long parameter lists, feature envy and data classes. Many of these smells have a large degree of subjectivity in their interpretation. For instance, how large is too large for a class? How do we justify (in the case of the feature envy smell) if one method is 'more interested' in another class than in that which it is defined? This leads us to the second challenge in performing a refactoring — identifying where to perform it.

Since many smells have a large degree of subjectivity or variety in their interpretation it is difficult to (reliably) automatically detect where to apply a refactoring. Much refactoring therefore relies upon the slow and tedious task of manually inspecting code. It would be beneficial to be able to reliably automatically detect where refactorings could be applied. To this effect we have identified a particular structure in a system's source code that can be automatically detected, and has a detrimental effect on the system's understandability, testability and reusability. The structure we have identified is long dependency cycles between classes in the system.

Long cycles among classes in Java programs create problems for developers because it is difficult to isolate any class in the cycle. Anyone wanting to understand any class in the cycle effectively has to understand every class in the cycle. This has implications for the cost of maintenance. Anyone wanting to test any class, effectively has to test every class. And anyone wanting to lift a class for reuse in another system, ends up having to lift every class in the cycle. This suggests software with cycles in the compilation dependency graph may be more costly to maintain than those without, which gives motivation for detecting and removing cycles.

Of course detecting and removing cycles would not be so interesting if they did not exist in "real software", or they were "mostly harmless". This leads into the contributions of this paper. One contribution is to show that cycles do exist in real software. We have done this by examining several widely-downloaded, open-source Java applications. In order to determine the prevalence of cycles we have built a tool to detect them — this is another contribution. Since we detect cycles from source code and not from byte code we have had to develop an algorithm for computing name bindings that is of little burden to implement, unlike a fully fledged Java compiler that by its very nature has to compute name-bindings and requires significant effort to implement — another contribution. The final contribution is showing how dependency cycles detected by our tool can be used as the starting point for refactoring.

The paper is organised as follows. In section 2 we motivate our work by discussing in more detail why cycles can create problems for software developers. We then discuss the literature related to our work in section 3. Section 4 presents the algorithm we use to create the compilation dependency graph. Section 5 discusses Jepends and section 6 shows the results of applying Jepends to a medium sized open source Java application. Section 7 discusses how the results of the analysis can be used to identify opportunities for refactoring, and finally section 8 presents our conclusions.

2 Motivation

Cycles in compilation dependency graphs (CDGs) have implications in understanding, testing, and reusing classes in the cycle. But are they really so bad? The simplest cycle is one involving two classes that depend on each other. It is very easy to find examples of such cycles – consider `java.lang.Class` and `java.lang.reflect.Method`, from the Java API for example. It is hard to argue this cycle is 'bad' because of the natural parent-child type relationship between a class and its methods. This relationship is represented at the source code level by `Class` providing a `Method[] getDeclaredMethods()` method and `Method` providing a `Class getDeclaringClass()` method. Breaking this cycle would involve terminating the parent's reference to its children or the children's reference to its parent, both of which are necessary relationships in order to provide usable `Method` and `Class` objects.

It would be tempting to simply declare 2-class cycles "good" and everything else bad, but we suspect "good" 3-class cycles can also be found, and so the question would then be at what size do cycles become "bad"? The 'necessary relationship' argument stated above is an appealing criteria, and may be a correct one, however it has the problem, from our point of view, that it is difficult to detect violations of it through mechanical analysis.

While it may be difficult to state categorically that a cycle of a certain size is bad, we would argue that it would be hard to argue that a large cycle, of size 50 for example, is something to be entirely happy with. We feel certain that it would be useful to know that cycles of that size (or larger) exist in our software, since that would provide a candidate for refactoring.

Large cycles in the CDG may indicate another problem. As we discuss in the next section, a number of authors have suggested that cycles of *subsystems* (groups of classes with coherent functionality) are bad. If we have a group of closely related classes (and so coherent functionality) then we would tend to want to understand, test, and reuse them as a unit. As we argued above, cycles within such classes may not be such a problem. However cycles between subsystems suggests that the subsystems are in fact not so coherent, and so again may indicate candidates for refactoring. The larger the cycle in a CDG, the larger the likelihood that the cycles cross subsystem boundaries. For example, if there is cycle of size 50, but all subsystems have fewer than 50 classes, then it must be that there is a cycle between subsystems.

Our goal then is to construct and analyse CDGs, and identify cycles, in particular large cycles.

3 Background

There has been a considerable amount of work done in analysing dependencies of different kinds. We mention only the most directly relevant here.

Graphs are a natural representation of computer programs well-suited for program analysis and transformation. Existing work in graph representations of programs is diverse. One dimension of this diversity is the context in which program entities are considered. Program entities may be considered dynamically — from the runtime state of the executing program, or statically — from the source code or an intermediate representation of it. Another dimension of work in graph representation of programs is the purpose for which the graph is used. Purposes include, but are not limited to, identifying violations of design heuristics, change propagation analysis, reverse engineering, reducing compilation time, and runtime performance optimisation. The work most relevant to this paper relates to identifying violations of design heuristics.

The earliest work in the area of runtime performance optimisation using graphs is by Kuck. Kuck introduces a *program dependency graph* in order to determine statements that can be executed in parallel in a (Fortran-like) program (Kuck, Muraoka & Chen 1972).

Program dependency graphs have also been used in order to analyse change propagation. The term 'ripple effect' is often used describe how a change can propagate (Black 2001). In essence, a change to the code of one module can have an effect on the data that is passed into other modules. This is of concern during software maintenance because a change to one module that may naively seem isolated could cause a regression fault in another.

In terms of reducing compilation time the graph representation typically comprises source files as vertices and compilation dependencies as directed edges. Yu et al. identify false dependencies as a cause of long compilation times and use a 'partitioning' operation on the graph in order to determine redundant `#include` statements (Yu, Dayani-Fard & Mylopoulos 2003). Cockerham uses a graph of dependencies amongst Ada source files in order to infer those files that can be compiled in parallel (Cockerham 1988). Assuming multiple processors are available for the compilation, its time is reduced.

Lague et al. generate a graph of dependencies between C/C++ source files through processing their *#include* statements (Lague, Leduc, Le Bon, Merlo & Dagenais 1998). This graph is used for reverse engineering in the sense that Lague et al. want to recover the layered architecture of the telecommunications system under study from its implementation (source files).

Several recent studies have profiled the overall characteristics of dependencies among classes in object oriented systems. Wheeldon et al. profiled the distributions of 5 different types of dependencies (e.g. inheritance, aggregation) in several large Java applications (Wheeldon & Counsell 2003). Marchesi et al. profiled the distributions of in-degrees and out-degrees for nodes in the class relationship graphs of 4 Smalltalk applications where the relationship took into account potential method invocations and superclasses (Marchesi, Pinna, Serra & Tuveri 2004). The authors of both studies found power laws in these distributions. Furthermore they speculated that these distributions are common across all large object oriented systems and that such distributions may be useful for predicting design complexity as a system grows and measuring the effects of refactorings on software quality. We also consider relationship graphs, however we concentrate on distributions related to the transitive closure of the relationships.

Work with compilation dependencies is usually associated with incremental compilation. Determining what needs to be recompiled when one source file is changed is non-trivial in Java. Lagorio has developed an algorithm for sound cascading recompilation in Java (Lagorio 2004) that deals with these issues. Lagorio's algorithm is sound in that its output is guaranteed to have the same effect as recompilation of the whole program. We have adapted Lagorio's algorithm to identify the relationships we are interested in.

Discussion in the literature of the consequences of dependency cycles is limited. Booch makes the observation that a CDG should be a directed acyclic graph as early as 1984, but provides no justification for it (Booch 1987, p.567). Szyperski also observes "...can introduce cyclic dependencies and threaten organizational structure" (Szyperski 1998, p.275).

In terms of dependency cycles between subsystems Riel (Riel 1996) provides a heuristic that states the model of the application should never be dependent on the user interface of that application. Presumably this heuristic aims to eliminate a dependency cycle between the model and view of the application.

Martin gives the Acyclic Dependency Principle (ADP), namely "the dependency structure between *packages* must be a directed acyclic graph" (our emphasis) where packages are defined similarly to subsystems but with an emphasis on reusability (Martin 1996). As we argued in the previous section, long cycles in the CDG may indicate that the ADP has been broken.

The most comprehensive discussion we found of dependency cycles among subsystems in object oriented software is given by Lakos. Lakos argues for the acyclic property on the basis that cyclic dependencies inhibit understanding, testing and reuse: "once two components are mutually dependent, it is necessary to understand both in order to fully understand either" (Lakos 1996, p.185).

Hautus has developed a tool to detect cycles between packages in Java applications and support removing them (Hautus 2002). His tool differs from ours in that it assumes classes are correctly organized into subsystems by the use of Java packages. The metrics his tool computes are far less comprehensive than ours and as far as we can tell his tool does not prioritize classes based on some notion of their need for refactoring.

4 Algorithm

We have developed an algorithm for inferring compilation dependencies between Java source files in an application. While this may seem at first thought trivial, it is not. As noted by Lagorio the rules for name binding (i.e. binding identifiers in Java source code to their corresponding program entities such as classes, methods, variables) are complicated. This is "because the dot notation is used to name many different kinds of things (types, packages, fields and so on), its semantics is context dependent and tricky" (Lagorio 2004).

Suppose we are presented with the dotted name (e.g., `a.b.C` in a Java source file. As stated in section 6.5 of the Java Language Specification the following happens to the name: "First, context causes a name syntactically to fall into one of six categories: PackageName, TypeName, ExpressionName, MethodName, PackageOrTypeName, or AmbiguousName. Second, a name that is initially classified by its context as an AmbiguousName or as a PackageOrTypeName is then reclassified to be a PackageName, TypeName, or ExpressionName. Third, the resulting category then dictates the final determination of the meaning of the name (or a compilation error if the name has no meaning)". There is a long set of rules for determining the name binding in each of the syntactic classifications. One option would be to implement all these rules in a program to infer dependencies. The other option is to find a heuristic based algorithm that is simpler to implement.

Fortunately there is a relatively simple (heuristic) algorithm for inferring dependencies between Java source files — it is described in Lagorio's work in sound, cascading recompilation in Java (Lagorio 2004). Lagorio's algorithm actually detects a superset of the actual dependencies of a source file. We have adapted Lagorio's algorithm so that it minimises the number of spurious dependencies detected, and ignores some compilation dependencies that are of little consequence to the developer's view of the system's class. The final output of our algorithm is a CDG whose vertices are source files and whose (directed) edges are compilation dependencies. The CDG is built up by processing the names, import statements and package declaration in each source file in order to determine a set of fully qualified type names to which that source file *may* refer. This set is subsequently used to infer dependencies between source files by comparing the type names in it to those declared by other source files in the application.

A simplified version of our algorithm can be expressed as follows: Let the source files in the application be denoted S_1, S_2, S_3, $\ldots S_n$. The output of the algorithm is an adjacency list representation of the program's compilation dependency graph of the form $S_i \to \mathcal{R}'_i$ where $\mathcal{R}'_i$ is the set of source files that S_i directly "refers-to", that is, those source files contain the declarations of types used in S_i.

Firstly consider names in Java that are used to refer to program entities such as methods, types, variables etc. A name can be simple, that is consist of a single identifier, or qualified, that is, consists of a sequence of 2 or more identifiers delimited by "." characters. We will express a name in the form $e_1.e_2.e_3.e_4.\ldots.e_k$ where e_j represents an identifier.

In order to construct $\mathcal{R}'_i$ we first compute $\mathcal{R}_i$ by combining, in a particular way, the names in the body of S_i that might refer to types with those appearing in the S_i's package declaration and import statements. $\mathcal{R}_i$ is the set of fully qualified class names to which S_i *may* refer. In Java fully qualified type names uniquely identify types within a program.

Let *onDemands*(S_i) be the set of names used in import-on-demand statements in S_i, as well as the package name that S_i belongs to. Import-on-demand statements are imports ending with a '.*'. Let *singleType*(S_i) be the set of names used in single-type-import statements in S_i. Single-type-import statements are imports that do not end with a '.*'. Let *body*(S_i) be the set of names that could refer to types in the body of S_i. Then

$$\begin{aligned}
\mathcal{R}_i^{\text{body}} &= \{e_1 \ldots e_j | e_1 \ldots e_j \ldots e_k \in \mathit{body}(S_i), \\
&\quad 1 \leq j \leq k\} \\
\mathcal{R}_i^{\text{ondemand}} &= \{e_1 \ldots e_j \ldots e_k | e_j \ldots e_k \in \mathcal{R}_i^{\text{body}}, \\
&\quad e_1 \ldots e_{j-1} \in \mathit{onDemands}(S_i)\} \\
\mathcal{R}_i^{\text{single}} &= \{e_1 \ldots e_j \ldots e_k | e_1 \ldots e_j \in \\
&\quad \mathit{singleType}(S_i), e_j \ldots e_k \in \mathcal{R}_i^{\text{body}} \vee \\
&\quad e_1 \ldots e_k \in \mathit{singleType}(S_i)\}
\end{aligned}$$

and so $\mathcal{R}_i = \mathcal{R}_i^{\text{single}} \cup \mathcal{R}_i^{\text{body}} \cup \mathcal{R}_i^{ondemand}$.

Let T be the set of all types declared in S_1, ..., S_n, then $\mathcal{R}'_i = \mathit{declaringSources}(\mathcal{R}_i \cap T)$ where *declaringSources* takes a set of type names and returns a set containing the source files in which the types are declared.

This presentation of the algorithm has been simplified by not taking into account all of the issues due to Java's rules for shadowed names, obscured names, and nested types. Lagorio discusses these issues in full detail (Lagorio 2004).

We illustrate the algorithm using the following source file.

```
 1: //file S1
 2: package a.b;
 3: import x.*;
 4: import y.Z;
 5: class MyClass {
 6:   private A a = new A();
 7:   public void doStuff() {
 8:      B b = new C();
 9:      a.exec();
10:      System.out.println();
11:   }
12: }
```

The different sets in the algorithm are:

$$\begin{aligned}
\mathit{body}(S_1) &= \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{System.out}\} \\
\mathcal{R}_1^{\text{body}} &= \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{System.out}, \texttt{System}\} \\
\mathit{onDemands}(S_1) &= \{\texttt{a.b}, \texttt{x}\} \\
\mathcal{R}_1^{\text{ondemand}} &= \{\texttt{a.b.A}, \texttt{a.b.B}, \texttt{a.b.C}, \\
&\quad \texttt{a.b.System.out}, \\
&\quad \texttt{a.b.System}, \texttt{x.A}, \texttt{x.B}, \texttt{x.C}, \\
&\quad \texttt{x.System.out}, \texttt{x.System}\} \\
\mathit{singleType}(S_1) &= \{\texttt{y.Z}\} \\
\mathcal{R}_1^{single} &= \{\texttt{y.Z}\} \\
\mathcal{R} &= \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{System.out}, \texttt{a.b.A}, \\
&\quad \texttt{a.b.B}, \texttt{a.b.C}, \texttt{a.b.System.out}, \\
&\quad \texttt{a.b.System}, \texttt{x.A}, \texttt{x.B}, \texttt{x.C}, \\
&\quad \texttt{x.System.out}, \texttt{x.System}, \texttt{y.Z}\}
\end{aligned}$$

It is worth noting that there were names in the body of the source that did not appear in *body*(S_1). Particularly `a` on line 6 does not appear because its context makes it a variable name, thus its name cannot refer to a type. Method declarations/calls such as `.exec()` (9), `doStuff()` (7) and `.println()` (10) do not appear because their context identifies them as methods. The `a` on line 9 does not appear because we can infer from the source file that it cannot refer to a type: it is in the scope of a declared field.

It is also worth noting that many of the names in each source file's $\mathcal{R}$ will identify types that are not declared in the application's other source files. Lagorio refers to these names as *ghost dependencies*. Since we are not interested in ghost dependencies we cull them from each source file's

$\mathcal{R}$ in order to get a new set $\mathcal{R}'$. To know which names to cull we build up a map from type to source file of all the types declared across all the source files in the application. This allows *declaringSources* to be computed.

The key difference between our algorithm and Lagorio's is in the construction of the refers to set, $\mathcal{R}$. We minimise the number of entries in $\mathcal{R}$ by resolving names to variables and types inside a source file where allowed by the Java Language Specification (JLS) (Gosling, Joy, Steele & Bracha 2000, Chapter 6). We remove ghost dependencies from $\mathcal{R}$. We do not add single-type-import statements to $\mathcal{R}$ whose types are not used in the body of the source file (contrary to the example above). While ignoring redundant single-type-imports is not sound in cascading recompilation, it is a minor concern in program analysis where we found it was causing many superfluous dependencies between source files.

4.1 Benefits

It is in many ways beneficial to infer dependencies from a system's source files and not its compiled code (i.e. byte code). While inferring a class's dependencies from its byte code is trivial (one can simply look at the fully qualified class names appearing in the class file's constant pool) the process of compiling source files to byte code is seldom straight-forward for a newly downloaded application. It can involve having to track down external libraries, modify build scripts for the local environment and so on. Furthermore if something is preventing the system from compiling (e.g. an unresolved reference or syntax error) then *no* dependencies can be computed. Downloading the application in its compiled form doesn't help much either because it then becomes difficult to determine which classes correspond to sources and which classes have originated from external libraries.

A major benefit of our algorithm is that it is specifies a simpler means of inferring dependencies between source files than the way in which a compiler goes about inferring these dependencies. For instance our algorithm is unconcerned with statement reachability checking, type checking and static context checking, whereas a compiler must perform these steps. As a consequence of the omission of such steps our algorithm should be faster at inferring dependencies between Java source files than a compiler. Even compared to the subsystem of a compiler whose purpose is to compute name bindings our algorithm is superior in that the compiler's subsystem is complicated to implement because it must implement the pages upon pages of rules discussed in section 6.5 of the Java Language Specification. Furthermore, again unlike a compiler, our algorithm does not require references to any external jar files used by an application in order to infer dependencies between sources.

Another benefit our algorithm is that it could be easily adapted to infer compilation dependencies between source files in other Java-like languages such as C#. The simplicity of the algorithm is such that it can be implemented in a few hundred lines of code assuming one starts with an off the shelf parser for the target language.

4.2 Limitations

While the algorithm we have described avoids much of the work performed by a compiler, which by its very nature has to infer dependencies, there are situations where it could detect spurious dependencies. Consider the following example in illustration of this.

```
package pack;
import x.*;
class Example {
   A a = new A();
}
```

Computing $\mathcal{R}$ for this source file yields {`pack.A`, `x.A`}. Assume that in the application's source files both types are declared. The JLS states that the types are resolved using the implicit package import in preference to import-on-demand statements (section 6.5.5) so in reality `Example` only depends on `pack.A`. Our algorithm (incorrectly) infers that `Example` depends on both `pack.A` and `x.A`. We expected this type of situation would be very rare. For a medium-sized Java application called Azureus we detected this situation, where two classes had the same simple name, and manually inspected all incidences of it in offending source files' texts. Of the 30 occurrences of conflicting names none caused erroneous references. In each case both classes were actually referenced in the source file's text: one using its fully qualified name and the other using its simple name in conjunction with a single-type-import.

Another way our algorithm could infer an erroneous reference is if a variable name was interpreted as a class name. This is analog to a potential problem stated in the JLS where a variable name could *obscure* a simple type name. Fortunately the convention of naming classes with an initial uppercase letter and naming variables with an initial lowercase letter minimizes this type of conflict (see JLS section 6.8). In all the systems we ran our tool on during its development we casually observed source files had obeyed this coding standard, almost certainly eliminating all erroneous references that could be generated in this way.

One final point to note is that in the general case our algorithm does not infer a direct dependency between a class that uses an inherited field or method, and the class that defines that field/method. Consider a class `A` using a field defined in its superclass's superclass `C`. Our algorithm detects an indirect dependency between `A` and `C` through `A`'s superclass. In this particular example a Java compiler would infer a direct dependency from `A` on `C`, and this would be written to `A`'s binary class file (see JLS 13.4.7). Briand et al's framework for measuring coupling more thoroughly addresses this issue (Briand, Daly & Wüst 1999).

5 Jepends

An implementation of the algorithm described in section 4 has a number of practical benefits in terms of the kinds of analysis we are interested in. In particular, it does not require that the source code be in a deployable (or even buildable) state. This avoids problems with source files not being available or organised incorrectly, dealing with external jar files or other subsystems, or configuration issues.

We have implemented the algorithm as part of our tool Jepends. Jepends uses the results of the algorithm to build up the compilation dependency graph, and then analyses the graph in various ways.

Jepends can compute a suite of sets for each of the application's source files: **Refers-to** — the $\mathcal{R}'$ set i.e., the other sources referred to directly by the names in the given source file; **Refers-to-tc** — the transitive closure of refers-to; **Referred-to-by** — the inverse of refers-to; **Referred-to-by-tc** — the transitive closure of referred-to-by; **Cycles-thru** — a subset of all simple cycles (no repeated vertices) that a given source file participates in. The size of the refers-to and referred-to-by sets give the out-degrees and in-degrees of the corresponding vertex in the compilation dependency graph. The transitive closure relations determine what source files either require or are required by a given file during the compilation process. Currently Jepends outputs dependency profiles as text files that can be imported into tools such as Excel for sorting, graphing and further analysis. Table 1 shows part of the output, in this case the top four classes when sorted by Cycles-thru. The **TC** columns are the transitive-closure

Class	Referred-to-by	TC	Refers-to	TC	Cycles-thru
org....config.COConfigurationManager	164	1003	5	1373	3280
org....config.impl.ConfigurationDefaults	3	1003	10	1373	3279
com.....defaultplugin.StartStopRulesDefaultPlugin	3	1003	38	1373	3275
org....logging.LGLogger	107	1003	4	1373	3274
...	...	...	...	...	...

Table 1: Part of the output by Jepends. Class names have been elided.

version of the column to the left. The fact that the numbers are the same for all classes in these columns is discussed in the next section.

The fact that Cycles-thru is a *subset* of all the simple cycles a given source file participates in requires further explanation. Efficiently finding all the simple cycles a given node in a directed graph participates in is a difficult problem (Alon, Yuster & Zwick 1994). One approach to finding all simple cycles (that is easily implemented in Java) is to find all simple paths between each pair of nodes the graph and determine which of these paths also correspond to a simple cycle. A simple path corresponds to a simple cycle if there exists an edge in the graph from the terminal node in the path to the initial node in the path. Several different paths can correspond to the same simple cycle and this is easily detected by checking that the paths contain the same nodes, and that these nodes occur in the same order (when they are arranged into a cycle).

Unfortunately finding all simple paths between all pairs of nodes is infeasible with respect to time for a graph of any decent size. Our approach is to keep track of all the simple cycles source files participate in that are encountered during the course of the depth first searches to construct the Refers-to-tc set of each node. In this regard Cycles-thru is a sample of the total cycles that pass through a node. More importantly it shows that a given node participates in *at least* this many simple cycles.

6 Results

In this section we demonstrate Jepends by using it on Azureus, an open-source application that provides peer-to-peer file sharing (Azureus 2005). Azureus is written in Java 1.4 and release 2.3.0.0 comprises 1913 Java source files with approximately 114000 lines of non-comment source statements. Azureus are uses the Standard Widget Toolkit for its user interface (like Eclipse), and has no automated unit test suite.

We came across Azureus because it frequently appears on Sourceforge's top 10 lists for number of downloads and development activity. Our end-user experience of Azureus is that it is easy to use, stable and feature-rich. This is atypical of our end-user experience with other peer-to-peer file-sharing applications. It also raises the question 'Is Azureus's internal design indicative of its positive end-user experience?'.

Figures 1 and 2 show the distribution of set sizes in the referred-to-by and refers-to relations. In the figures, the x-axis is the size of the sets and the y-axis is the number of classes that have a given sized set. So figure 1 says that about 1800 classes have refers-to-by sets of size between 0 and 19. Both distributions show that small values are extremely common whereas large values are very rare. This is reminiscent of the power law relationships found by Marchesi et al (Marchesi et al. 2004).

Figures 3 and 4 respectively show the distributions of the set sizes for refers-to-tc and referred-to-by-tc. The distributions in figures 3 and 4 are of particular interest. Both distributions show two distinct clusters: from 0-99 and 1000-1199 for referred-to-by-tc distribution, and from 0-99 and 1300-1499 in the refers-to-tc distribution. These seem to be very odd distributions — in the case of referred-to-by-tc, this says that between 1000 and 1199 classes depend (transitively) on nearly 1400 other classes.

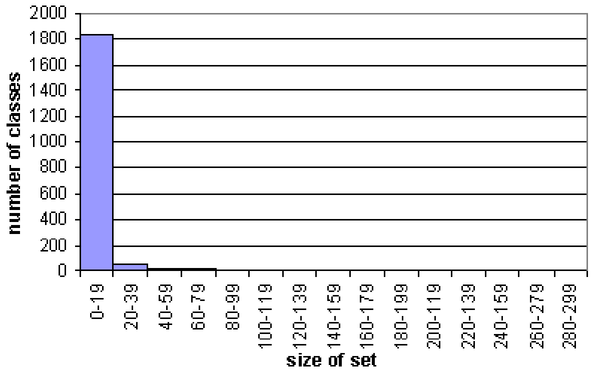


Figure 1: Azureus' referred-to-by distribution

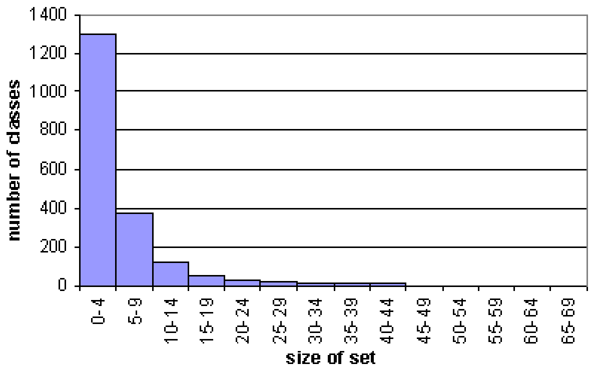


Figure 2: Azureus' refers-to distribution

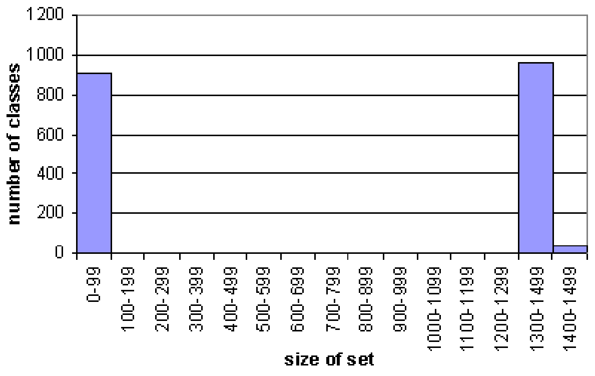


Figure 3: Azureus' refers-to-tc distribution

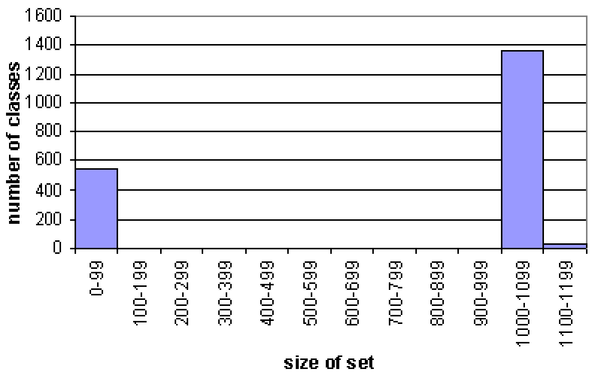


Figure 4: Azureus' referred-to-by-tc distribution

Furthermore, the distributions indicate no classes depend on (for example) 500 other classes. It is very much that classes depend on only a few classes (fewer than 100) or most of the classes.

The question then is, is this distribution somehow characteristic of all applications, or somehow peculiar to Azureus. If it is peculiar to Azureus, then the presence of such distributions may tell us something about the nature of Azureus' design. We used our tool to examine the distribution of these relations in other systems such as Tomcat 5.5.9, Eclipse 3.0 and Netbeans 3.6 and found some clustering, but overall large valued clusters were less common than small valued clusters as exemplified by Tomcat's refers-to-tc distribution in Figure 5.

Now the question is, why does Azureus have such odd distributions? Is it just some particular characteristic of the application that is not related to the design, or is it indicative of some, possibly bad, design characteristic?

In fact, such distributions indicate the possible presence of long cycles. To see this, consider the distribution in Figure 3. The right-hand cluster indicates that of the approximately 1900 source files in Azureus, about 1000 of them depend (either directly or transitively) on 1300 or more other source files. The left-hand cluster indicates that the remaining 900 or so source files in the application depend on between 0 and 99 other source files. In fact the 900 source files in the left-hand cluster cannot depend on any in the right-hand cluster because of the transitivity. If a source file in the left-hand cluster depended on one in the right-hand cluster, it would depend on all the source files the latter depended on, which we know is 1300 or more, and so that source file should have been in the right-hand cluster.

Files in the right-hand cluster can refer to files in the left-hand cluster, but since there are at most 900 in the left-hand cluster that means *every* file in the right-hand cluster must refer to at least *one other* file in the right-hand cluster, meaning there must be cycles within the right-hand cluster. The length of the cycles depends on the internal structure of the CDG, however we get hints by looking at the raw output of Jepends as shown in table 1. As noted earlier, the values of the **TC** columns for the classes shown are all the same. This means that with transitive closure they all have the same set of classes that they depend on or are depended on, which could be explained by all of the classes belonging to a cycle.

It was the appearance of the odd distributions for Azureus compilation dependencies and other applications that led to our interest in cycles, and the introduction of cycle profiling to Jepends. If we use Jepends to profile the distribution of lengths of unique simple cycles we get the graph as shown in Figure 6. Note that because vertices in the graph can participate in more than one unique cycle, the sum of the frequencies is greater than the number of source files. The graph shows that there are a large number of long cycles in Azureus. Indeed 75% of the cycles in involve more than 50 nodes. Now the question is how we can use this information to identify possibilities for refactoring, which we discuss in the next section.

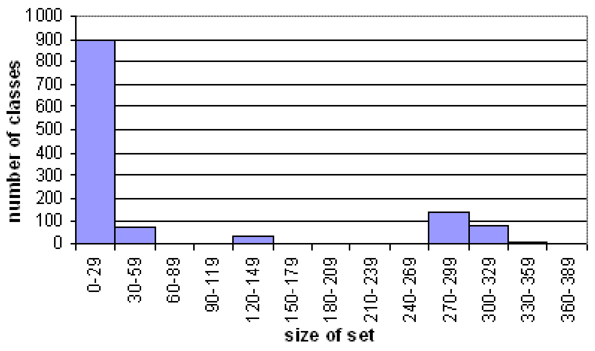


Figure 5: Tomcat's refers-to-tc distribution

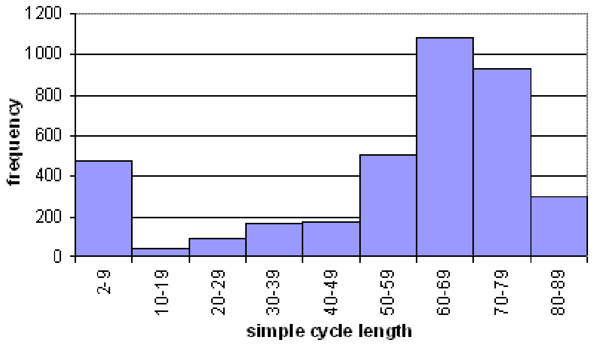


Figure 6: Azureus' simple cycle length distribution

7 Refactoring

In this section we will explain how the analysis by Jepends can be used to indicate starting points for refactoring and measure the effect a refactoring on dependencies. The data in table 1 comes from Azureus and, as mentioned earlier, shows the top 4 classes when files are sorted by the number of cycles in which they participate.

Based on this data, we surmise that breaking the cycles through `COConfigurationManager` may greatly reduce the total number of (long) cycles in the system. A technique for breaking all cycles through `COConfigurationManager` would be to extract an interface from it and replace all existing references to its implementation with the extracted interface. In order to avoid a dependency on the interface's implementation, we would have to further refactor the classes referencing `COConfigurationManager` not to create a new instance of, or statically depend on, its implementation.

While the 'extract interface' refactoring would definitely reduce the number of cycles in a system the overall effect on design quality by repeatedly performing this refactoring is dubious. The repeated use of the refactoring would dramatically increase the total number of source files in the system and the existence of the interfaces defined in these files would be justified on the basis of reducing cycles alone.

A refactoring whose justification can be more strongly argued is more subtly indicated by the data in table 1. The name `COConfigurationManager` suggests that its class is involved in something to do with configuration, potentially belonging to a configuration subsystem. Upon inspection of this class's source we find that it is the Facade into the configuration subsystem. The configuration subsystem is responsible for loading and saving user configurable parameters used throughout Azureus's code (e.g. the directory to which files download, and the maximum download and upload rates). These parameters are saved to flat text files so they can remain persistent between executions of Azureus.

It is hard to believe that functionality as primitive as saving and reading properties from disk should transitively depend on 1373 other classes. We think that in a better design for the configuration subsystem would depend only on the threading subsystem and the logging subsystem. These two subsystems are themselves primitive and probably should not depend on any other source files in Azureus. By a brief code inspection we identified 5 classes relating to threading: `AEMonitor`, `AEMonSem`, `AERunnable`, `AESemaphore`, `AEThread`. These classes were mixed up with other utility-type classes in the `org.gudy.azureus2.core3.util` package. In the logging subsystem (comprising its own package) we found 4 source files: `ILoggerListener`, `LGAlert-Listener`, `LGLogger`, `LGLoggerImpl`. Since the configuration subsystem (again in its own package) contains 13 files we would expect `COConfigurationManager` to transitively refer-to no more than 22 other files (=5+4+13). In any case this is an order of magnitude less than its current 1373.

The point of this discussion is to support our claim that the analysis provided by Jepends provided very valuable insight into the current design of Azureus, and so provided a useful starting point for the refactoring process.

8 Conclusions

In this paper we have discussed how data from the automated analysis of source code can be used to identify opportunities for refactoring. We have developed an algorithm based on work by Lagorio on incremental compilation, that allows compilation dependency graphs to be created for an application. We have implemented this algorithm in Jepends, which also analyses the resulting graph. We have provided canonical examples of refactorings indicated by running Jepends over the open-source Java application Azureus.

Many characteristics of the distributions of dependencies we found in Azureus' source are not unique to Azureus. We have seen similar distributions in a number of other applications that we have analysed. However we have also seen different distributions (such as Tomcat's). The fact that different distributions are possible suggest that it may be possible to get a sense of the quality of the design by profiling these distributions. We are completing the analysis of these other applications to better understand the relationship between different profiles and design quality.

Jepends and the algorithm it is based on are Java specific. However the principles behind their development are not language specific. We intend to widen the scope of Jepends in order to carry out large-scale studies on commercial software.

References

Alon, N., Yuster, R. & Zwick, U. (1994), Finding and counting given length cycles, *in* 'ESA '94: Proceedings of the Second Annual European Symposium on Algorithms', Springer-Verlag, London, UK, pp. 354–364.

Azureus (2005), 'Azureus project page', `http://azureus.sourceforge.net`. Sourceforge project page for Azureus.

Black, S. (2001), 'Computing ripple effect for software maintenance', *Journal of Software Maintenance* **13**(4), 263–279.

Booch, G. (1987), *Software components with Ada: Structures, tools, and subsystems*, Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

Briand, L. C., Daly, J. W. & Wüst, J. K. (1999), 'A unified framework for coupling measurement in object-oriented systems', *IEEE Transactions on Software Engineering* **25**(1), 91–121.

Cockerham, B. (1988), Parallel compilation of Ada units, *in* 'TRI-Ada '88: Proceedings of the conference on TRI-Ada '88', ACM Press, New York, NY, USA, pp. 147–164.

Fowler, M. (1999), *Refactoring: improving the design of existing code*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Gosling, J., Joy, B., Steele, G. & Bracha, G. (2000), *The Java(tm) Language Specification*, Addison-Wesley.

Hautus, E. (2002), Improving java software through package structure analysis, *in* 'The 6th IASTED International Conference Software Engineering and Applications'.

Kuck, D., Muraoka, Y. & Chen, S. (1972), 'On the number of operations simultaneously executable in fortran-like programs and their resulting speedup', *IEEE Transactions on Computers* **21**(12).

Lagorio, G. (2004), 'Capturing ghost dependencies in java sources', *Journal of Object Technology* **3**(11), 77–95.
URL: *`http://www.jot.fm/issues/issue_2004_12/article4`*

Lague, B., Leduc, C., Le Bon, A., Merlo, E. & Dagenais, M. (1998), An analysis framework for understanding layered software architectures, *in* 'IWPC '98: Proceedings of the 6th International Workshop on Program Comprehension', pp. 24–26.

Lakos, J. (1996), *Large-scale C++ software design*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Marchesi, M., Pinna, S., Serra, N. & Tuveri, S. (2004), Power laws in smalltalk, *in* 'ESUG Conference 2004 Research Track'.
URL: *`http://www.iam.unibe.ch/publikationen/techreports/2004/iam-04-008/file/at_download`*

Martin, R. C. (1996), Granularity, *in* 'The C++ Report'. `http://www.objectmentor.com/resources/articles/granularity.pdf`.

Riel, A. J. (1996), *Object-Oriented Design Heuristics*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Szyperski, C. (1998), *Component software: beyond object-oriented programming*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Wheeldon, R. & Counsell, S. (2003), Power law distributions in class relationships, *in* 'Third IEEE International Workshop on Source Code Analysis and Manipulation', p. 45.

Yu, Y., Dayani-Fard, H. & Mylopoulos, J. (2003), Removing false code dependencies to speedup software build processes, *in* 'CASCON '03: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research', IBM Press, pp. 343–352.

Unsupervised band removal leading to improved classification accuracy of hyperspectral images

R. Ian Faulconbridge, Mark R. Pickering and Michael J. Ryan
School of Information Technology and Electrical Engineering
UNSW@ADFA
Campbell, ACT, 2600
i.faulconbridge@adfa.edu.au

Abstract

Remotely-sensed images of the earth's surface are used across a wide range of industries and applications including agriculture, mining, defence, geography and geology, to name but a few. Hyperspectral sensors produce these images by providing reflectance data from the earth's surface over a broad range of wavelengths or bands. Some of the bands suffer from a low signal-to-noise ratio (SNR) and do not contribute to the subsequent classification of pixels within the hyperspectral image. Users of hyperspectral images typically become familiar with individual images or sensors and often manually omit these bands before classification.

We propose a process that automatically determines the spectral bands that may not contribute to classification and removes these bands from the image. Removal of these bands improves the classification performance of a well-researched hyperspectral test image by over 10% whilst reducing the size of the image from a data storage perspective by almost 30%. The process does not rely on prior knowledge of the sensor, the image or the phenomenology causing the SNR problem.

In future work, we aim to develop compression algorithms that incorporate this process to achieve satisfactory compression ratios whilst maintaining acceptable classification accuracies.

Keywords: Hyperspectral, classification, SNR.

1 Introduction and Context

Modern hyperspectral sensors are typically passive, optical sensors that record spectral radiance from the earth's surface at fine spatial resolutions. These sensors are capable of recording data across a wide range of wavelengths resulting in a large number of spectral channels or bands. The large number of contiguous spectral bands associated with each image allows end-users to differentiate and classify more accurately the covering of the earth's surface.

By matching the given spectral signatures to those of the pixels in the image, the classification process is able to determine the class or type of pixels in the image. Users typically "train" the algorithm using areas within the image that they know to be of a certain pixel type and then allow the algorithm to classify the rest of the image automatically. The data used to train the classification algorithm is called "ground truth" data.

However, the spectral and spatial resolution of hyperspectral images result in large data volumes causing storage, transmission and archival challenges. Additionally, the performance of the individual bands or channels varies from channel to channel and sensor to sensor resulting in variable signal-to-noise ratios (SNRs) across the channels.

Analysts and researchers in the remote sensing field typically become familiar with the individual sensor or data they are using and manually remove or ignore bands with low SNRs (see for example Shah, Watanachaturaporn, Arora and Varshney 2003, and Tsai and Philpot 2002). Removing these bands invariably improves the quality of the data and enhances subsequent processing, such as classification.

This paper proposes a process to identify the low SNR spectral bands and remove automatically these bands from the image during a pre-processing stage. The process is purely statistical and does not rely on knowledge of the phenomenology causing the SNR problem. The context of the proposed pre-processing stage is shown in Figure 1.

Figure 1 shows the original hyperspectral data moving through a *pre-processing* stage which is where the proposed process is implemented. Work on the *data compression* stage is continuing and is not part of this paper but is shown to assist with context. The compressed data is then *transmitted* or *archived*. Users of the data then *decompress* the data, *manipulate* it as required before *classifying* the data. It is during *end-user manipulation* that analysts currently (manually) select and omit low-SNR spectral bands.

By automating the removal of these bands during the *pre-processing* stage, the process becomes independent of sensor, data set and user experience allowing it to be employed even if the user is unfamiliar with the sensor and its image characteristics. It also caters for changes in sensor performance over time. The process improves the subsequent classification accuracy of a test image and reduces the image entropy (entropy being a probabilistic

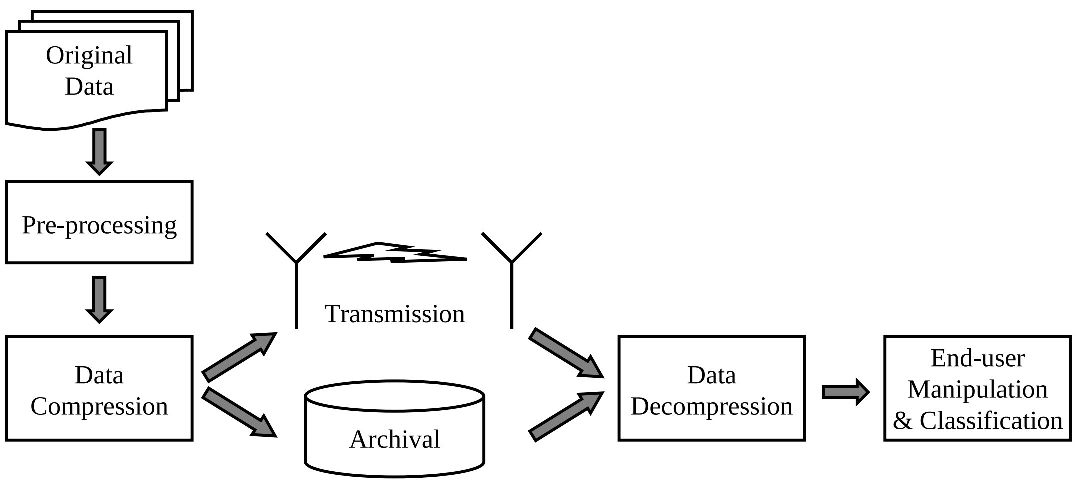


Figure 1: Image pre-processing in context with overall process

measure of the number of bits per data point required to encode and store the image).

The image used in this work is the 92AV3C image from the NASA Jet Propulsion Laboratory (JPL) sensor called the “Airborne Visible and Infrared Imaging Spectrometer” (AVIRIS). The AVIRIS sensor is typicallyflown on either an ER-2 jet aircraft which is a modified U2, or a Twin Otter turboprop (see “AVIRIS – General Overview”). Each pixel in an AVIRIS image contains 224 spectral channels from 400 to 2500 nm. The 92AV3C image consists of 145 by 145 pixels at 224 spectral bands forming a “data cube” of over 4.5 million data points. Four of the 224 spectral bands in the 92AV3C image contain zero values leaving a total of 220 non-zero bands (Shah, Arora, Robila and Varshney 2002).

This image was chosen for this work because it is freely available courtesy of Purdue University (see “92AV3C”), includes the necessary ground truth reference data needed for classification and accuracy assessments, and has become somewhat of an industry standard test image.

2 Method

The method used to identify the bands for removal is an extension of previous work which investigated the effect of compression-induced distortion on the classification of the 92AV3C image (Faulconbridge, Pickering, Ryan and Jia 2003). This worked showed that some bands are more important than others in the 92AV3C image when it comes to classification performance. The process proposed here aims to identify the least important bands contained in the image.

The first stage in our process is to normalise the image using image mean and standard deviation to produce a data cube with zero mean and unit variance. Normalising the image prevents bands with large spectral returns from dominating bands with smaller returns during subsequent processing. We then perform an unsupervised clustering of the image to group the pixels into statistically like groups based on their spectral response. We use a simple k-means clustering algorithm to do this (for an explanation of k-means clustering, see Anderberg 1973). Each pixel in the image is assigned to the closest k-means vector using Euclidean distance as the measure of “closeness”. Methods exist to determine the number of clusters required for use in different applications (Hardy and Lallemand 2002). Based on these methods and our previous work, ten clusters are used in our process. Each cluster can be characterised by its centroid (or mean vector) and variance.

In this paper, we are looking for spectral locations where all of the clusters are very close to one another. We use a measure of statistical distance (or separability) between clusters to determine these locations. Our thesis is that spectral locations where there is very little statistical separation between clusters will not be useful in differentiating pixels of different class during user classification. These are the locations we mark for removal. On the other hand, locations where the clusters are very distinct from one another will contribute to differentiating pixels during classification and these are the locations we retain.

Using first-order statistics only, such as cluster means, to measure separability, has been shown to be weak in a number of regards (Sweet 2003, and Shah, Arora, Robila and Varshney 2002). To overcome the potential problems associated with first order statistics, we use a well-established measure called the Bhattacharyya distance (Richards and Jia 1999) to quantify the separation between clusters as follows:

$$B_{ij} = \frac{1}{8}\left(C_i - C_j\right)^T \left(\frac{\Sigma_i + \Sigma_j}{2}\right)^{-1} \left(C_i - C_j\right) + \frac{1}{2}\ln\left(\frac{\left|\left(\Sigma_i + \Sigma_j\right)/2\right|}{\left|\Sigma_i\right|^{1/2}\left|\Sigma_j\right|^{1/2}}\right) \qquad (1)$$

where B_{ij} is the Bhattacharyya distance between cluster i and cluster j, C_i is the centroid of cluster i, and Σ_i is the covariance matrix associated with cluster i.

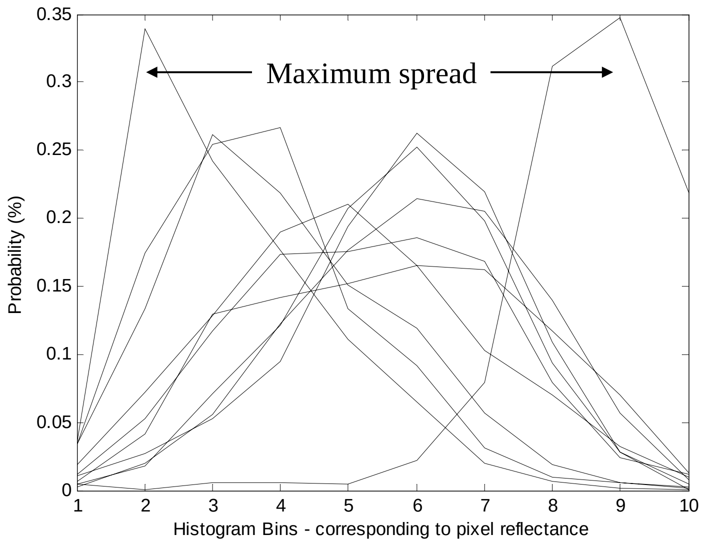


Figure 2: Histogram showing 10 clusters in a well-separated spectral band (band 50 of 220)

Scalar measurements such as the Bhattacharyya distance in Equation 1 provide no indication of where (spectrally) the differences between the two clusters lie. As we are looking to appreciate the separability of each cluster (from each other cluster) as a function of spectral band, we calculate a Bhattacharyya distance between each cluster for each spectral band using the following modified expression:

$$B_{ij,n} = \frac{\left(C_{i,n} - C_{j,n}\right)^2}{4\left(\sigma_{i,n}^2 + \sigma_{j,n}^2\right)} + \frac{1}{2}\ln\left(\frac{\sigma_{i,n}^2 + \sigma_{j,n}^2}{2\sigma_{i,n}\sigma_{j,n}}\right) \qquad (2)$$

where $B_{ij,n}$ is the Bhattacharyya distance between cluster i and cluster j at band n, $C_{i,n}$ is the centroid of cluster i at band n and $\sigma_{i,n}$ is the standard deviation of cluster i at band n.

Using Equation 2 gives us n 1-dimensional Bhattacharyya distances for each cluster pair, measuring the distance between a given cluster and all of its neighbours as a function of spectral band (in the case of the 92AV3C image, n equals 220 as explained earlier).

As a next step, we look for spectral locations where the separation between a cluster and its most-distant neighbour is quite small. In other words, we are looking for spectral locations where the maximum Bhattacharyya distance or “spread” between all clusters is small.

Calculating the separation of the clusters in this way is computationally inexpensive because each cluster is simply represented by an n-dimensional mean vector and an n-dimensional standard deviation vector. Calculations are not performed on each pixel in the image. Additionally, the calculations are performed during pre-processing not during end-user classification which further reduces the impact of the technique on the user and their computational resources.

In a majority of the spectral bands, there is reasonably clear separation between the individual clusters. Figure 2 shows a histogram using actual data from the 92AVC3 image at one of the well-separated bands. The concept of maximum spread between these clusters is marked on Figure 2 to illustrate the idea of separation. These locations correspond to large Bhattacharyya distances and it is bands like the one shown that facilitate accurate classification processing.

However in other spectral locations, there is no appreciable separation (see Figure 3) corresponding to small maximum Bhattacharyya distances. These locations therefore provide little or no assistance to classification algorithms in differentiating and classifying pixels.

We identify the bands where the maximum Bhattacharyya distance is less than a given threshold (B_T) and mark these bands for removal. The thresholds used are discussed in Section 3. These locations coincide with low-SNR bands in the image and we remove the bands automatically during the pre-processing stage in Figure 1. Removing the bands reduces the entropy (or size) of the image and improves the classification process as detailed in the Section 3.

3 Results

Our experimental process starts by measuring the entropy and classification accuracies associated with the original 92AV3C image. These results become the baseline against which we compare the results of our subsequent pre-processing. We used Multispec© which is a freely available application widely used in remote sensing education, research and practice (see “Purdue/LARS Multispec©”) to perform the classification. We use maximum likelihood “leave one out” classification of the training fields in the 92AV3C image. “Leave one out” classification involves classification of each pixel in the training fields based on class statistics calculated using the remaining pixels in that class and provides an unbiased assessment of classification accuracy compared to the re-substitution method which produces optimistic estimates of accuracy (Landgrebe and Biehl 2001).

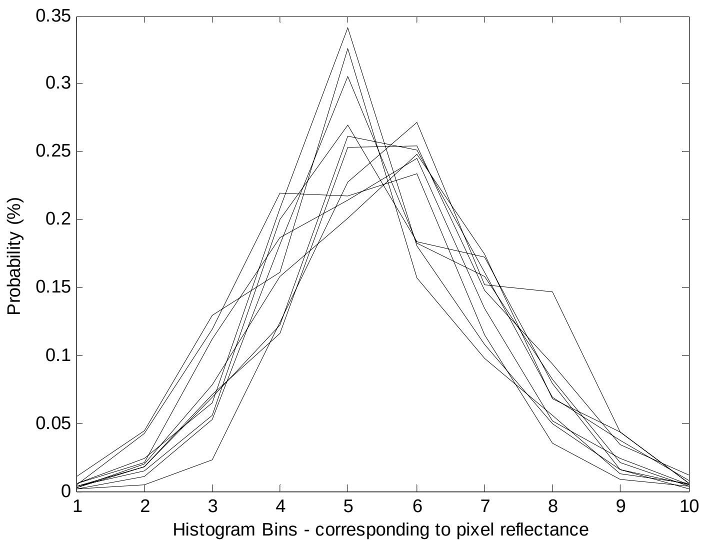


Figure 3: Histogram showing 10 clusters in a poorly-separated spectral band (band 105 of 220)

Although the ground truth data for 92AV3C shows that the image contains 16 different classes of pixel, only 11 classes were used as 5 of the classes contain less than 220 pixels in the training set and cannot be used for classification by Mulitspec© using maximum likelihood classification (as the number of training samples is less than the number of bands in the image). The original 92AV3C image has an entropy of 10.67 bits/data point and an overall class accuracy of 70.5%.

We then start to remove bands from the 92AV3C image where the maximum Bhattacharyya distance between clusters is less than a given threshold and classify the resulting image using Multispec©. Table 1 shows the bands that our process removes from 92AV3C as a function of Bhattacharyya distance threshold. Table 1 also notes the reduction in entropy (or size) of the resultant image and the overall classification accuracy following classification.

Table 1 shows that as we increase the Bhattacharyya threshold (T_B), we effectively remove more and more of the bands in the image. It is important to emphasise that the removal process described herein is performed without *a priori* knowledge of either the AVIRIS sensor or 92AV3C image and is therefore performed during pre-processing in Figure 1.

The results are summarised in Figure 4 which shows overall classification accuracy versus the Bhattacharyya threshold.

Figure 4 shows that as T_B is increased to 8 (and corresponding bands are removed), the overall classification accuracy of the image improves from 70.5% to 80.7%. Once T_B is increased beyond 8, we start to remove bands containing useful information and the overall classification accuracy therefore starts to fall away.

Table 2 shows the classification accuracy for each of the 11 classes in the original image compared to the accuracies achieved following the band removal process when a T_B of 8 is used. In general, there have been significant improvements in the classification accuracy of individual classes including improvements of over 20% for *Corn-min*, *Grass/pasture*, *Soybeans-no till*, and *Buildings*.

Soybeans-min and *Woods* are exceptions in that they suffer a minor reductions in classification accuracy but remain classified at close to original accuracies. We hope to investigate these exceptions in future work.

4 Conclusions and Future Work

During our investigations into the compression of the 92AV3C AVIRIS image, we have developed a technique that automatically identifies and removes low SNR bands from the image. The benefits of this technique are:

1. the user does not need to be familiar with the image or the sensor in order to remove those bands that do not contribute to classification accuracy,
2. the removal of the bands improves the classification performance from the image, and
3. the entropy (or size) of the image is reduced because of the reduction in data contained therein, resulting in faster processing times and reduced storage requirements.

In the case of the 92AV3C image, we were able to improve the overall classification accuracy of the image from 70.5% to 80.7% whilst reducing the entropy of the image from 10.67 bits/data point to 7.59 bits/data point.

We intend to test this technique further using alternative measures of separability and clustering techniques, and using different hyperspectral images. We also intend to develop compression algorithms that incorporate this pre-processing to achieve satisfactory compression ratios whilst maintaining acceptable classification accuracies

.

Bhattacharyya Threshold (T_B)	**Number of bands removed**	**Bands removed**	**Entropy (bits/data point)**	**Classification accuracy (%)**
0	0	-	10.67	70.5
1	23	1, 104-109, 150-163, 219-220	9.80	75.4
2	27	1-2, 103-109, 149-164, 219-220	9.64	75.9
3	31	1-3, 95-96, 103-109, 149-164, 218-220	9.48	76.9
4	52	1-4, 36-37, 80-99, 103-109, 149-164, 218-220	8.61	79.5
5	57	1-5, 36-37, 78-100, 103-110, 149-164, 218-220	8.40	79.8
6	63	1-7, 35-37, 78-110, 149-165, 218-220	8.19	80.0
7	67	1-9, 35-38, 77-110, 149-165, 218-220	8.01	80.3
8	76	1-17, 35-38, 77-110, 149-165, 217-220	7.59	80.7
9	87	1-18, 34-39, 60-65, 75-110, 149-165, 217-220	7.04	79.7
10	102	1-19, 34-44, 60-110, 149-165, 217-220	6.28	79.6
11	121	1-21, 34-111, 149-165, 215, 217-220	5.45	78.0
12	124	1-23, 34-111, 149-165, 215-220	5.31	77.4
13	129	1-25, 31, 33-112, 149-165, 215-220	5.07	74.8
14	131	1-26, 31, 33-112, 148-165, 215-220	4.97	74.6
15	137	1-112, 148-165, 214-220	4.72	71.1

Table 1: Progressive results as the Bhattacharyya distance is increased

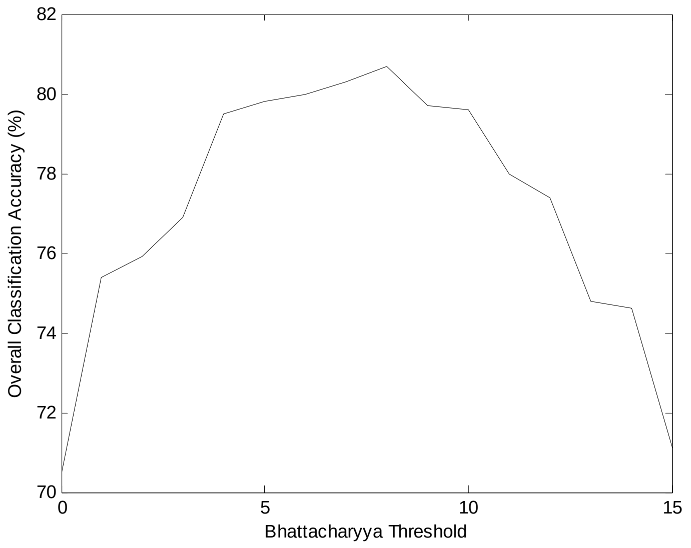


Figure 4: Overall classification accuracy vs Bhattacharyya threshold

Class	Classification Accuracy before band removal (%)	Classification Accuracy after band removal (%)	Improvement (%)
Corn-no till	78.5	82.8	4.3
Corn-min	40.6	70.1	29.5
Corn	0	2.5	2.5
Grass/pasture	58.2	90.3	32.1
Grass/trees	95.2	99.3	4.1
Hay-windrowed	99.6	100	0.4
Soybeans-no till	45.3	70.0	24.7
Soybeans-min	96.6	91.9	-4.7
Soybeans-clean	26.8	72.2	45.4
Woods	99.8	99.5	-0.3
Buildings etc	20.5	60.6	40.1

Table 2: Classification for each class before and after pre-processing (T_B=8)

5 References

Shah, C. A., Watanachaturaporn, P., Arora, M. K. and Varshney, P. K. (2003): Some Recent Results on Hyperspectral Image Classification, *IEEE Workshop on Advances in Techniques for Analysis of Remotely Sensed Data*, Greenbelt, MD, USA.

Tsai, F. and Philpot, W.D. (2002): A Derivative-Aided Hyperspectral Image Analysis System for Land-Cover Classification, *IEEE Transactions on Geoscience and Remote Sensing*, **40**:2:416-425.

AVIRIS – General Overview, NASA http://aviris.jpl.nasa.gov/. Accessed 27 Jul 2005.

Shah, C.A., Arora, M.K., Robila, S.A. and Varshney, P.K. (2002): ICA Mixture Model based Unsupervised Classification of Hyperspectral Imagery, *31st Applied Imagery Pattern Recognition Workshop*, Washington DC, USA.

92AV3C: Hyperspectral Test Image Data Pack, Purdue University ftp://ftp.ecn.purdue.edu/biehl/MultiSpec/92AV3C. Accessed 31 July 2005.

Faulconbridge, R.I., Pickering, M.R., Ryan, M.J. and Jia, X. (2003): A New Approach to Controlling Compression-Induced Distortion of Hyperspectral Images, *Proceedings of The International Geoscience and Remote Sensing Society Conference*, Toulouse, France, **III**:1830-1832.

Anderberg, M.R. (1973), Cluster Analysis for Applications, Academic Press, New York.

Hardy, A. and Lallemand, P. (2002): Determination of the Number of Clusters for Symbolic Objects Described by Interval Variables. In *Classification, Clustering and Data Analysis*, 311-316, Springer-Verlag, Berlin.

Sweet, J.N. (2003): The spectral similarity scale and its application to the classification of hyperspectral remote sensing data, *IEEE Workshop on Advances in Techniques for Analysis of Remotely Sensed Data*, 92-99, Greenbelt, MD, USA.

Richards, J.A. and Jia, X. (1999): Remote Sensing Digital Image Analysis, Springer-Verlag, Berlin.

Purdue/LARS Multispec©, Remote Sensing Freeware Software, Purdue Research Foundation, http://dynamo.ecn.purdue.edu/~biehl/Multispec©/. Accessed 27 Jul 2005.

Landgrebe, D. and Biehl, L. (2001): An Introduction to Multispec, p133, Purdue Research Foundation, Indiana, USA.

On Compensating the Mel-Frequency Cepstral Coefficients for Noisy Speech Recognition

Eric H. C. Choi

Interfaces, Machines and Graphic Environments (IMAGEN)
National ICT Australia
Locked Bag 9013, Alexandria, NSW 1435, Sydney, Australia

Eric.Choi@nicta.com.au

Abstract

This paper describes a novel noise-robust automatic speech recognition (ASR) front-end that employs a combination of Mel-filterbank output compensation and cumulative distribution mapping of cepstral coefficients with truncated Gaussian distribution. Recognition experiments on the Aurora II connected digits database reveal that the proposed front-end achieves an average digit recognition accuracy of 84.92% for a model set trained from clean speech data. Compared with the ETSI standard Mel-cepstral front-end, the proposed front-end is found to obtain a relative error rate reduction of around 61%. Moreover, the proposed front-end can provide comparable recognition accuracy with the ETSI advanced front-end, at less than half the computation load.

Keywords: Speech recognition, noise robustness, front-end processing, Mel-frequency cepstral coefficient.

1 Introduction

The proliferation of handheld computing devices has been the driving force behind the growing needs of more usable and natural user interfaces for ubiquitous computing. Traditional user interfaces based on the use of keyboard and mouse will not fulfill the needs of these mobile users. Automatic speech recognition (ASR) plays a critical role in providing more user-friendly user interfaces for these handheld devices. However since a handheld device can be used anywhere and in different environments, the design of a speech recognition system must take the potential noisy acoustic environments into consideration.

Automatic speech recognition basically consists of two stages (Rabiner and Juang 1993). The first stage, known as front-end processing or feature extraction, is aimed at extracting a time sequence of feature vectors which represents the temporal evolution of the spectral characteristics of a speech signal. The second stage is a pattern matching process where actual search is carried out to decode the spoken words by matching the sequence of feature vectors against the acoustic and language models stored in the recogniser. In state-of-the-art ASR systems, the features extracted in front-end processing are typically Mel-frequency cepstral coefficients (MFCC) and the pattern matching is mostly based on hidden Markov modelling (HMM) which requires relevant speech samples to train the acoustic models beforehand.

State-of-the-art ASR systems work pretty well if the training and usage conditions are similar and reasonably benign. However, under the influence of noise, these systems begin to degrade and their accuracies may become unacceptably low in severe environments (Deng and Huang 2004). To remedy this noise robustness issue in ASR due to the static nature of the HMM parameters once trained, various adaptive techniques have been proposed. A common theme of these techniques is the utilisation of some form of compensation to account for the effects of noise on the speech characteristics. In general, a compensation technique can be applied in the signal, feature or model space to reduce mismatch between training and usage conditions (Huang et al. 2001).

Signal-space methods, e.g. (Ephraim 1992), typically try to enhance a noisy speech signal by improving its signal-to-noise ratio (SNR). However, increase in SNR does not always contribute to improvement in recognition accuracy. Feature-space methods, e.g. (Hermansky 1990), try to derive some kind of feature representation that is potentially invariant to the change in environmental noise conditions. This is often achieved by incorporating some aspects of human auditory modelling. Alternatively, some other feature-space methods (Sankar and Lee 1996; Choi 2004) try to understand and compensate the effects of noise on a speech representation and correspondingly reduce the mismatch. Model-space methods, e.g. (Yao et al. 2001; Zhang and Furui 2004), try to adjust the recognition model parameters to incorporate the effects of noise on the acoustic models.

A few standards for ASR feature extraction are available from the European Telecommunications Standards Institute (ETSI). The standard WI007 Mel-cepstral front-end (ETSI 2000) covers the processing of speech signal into MFCCs. As this basic front-end is not that robust for noisy speech recognition, another standard that is more appropriate for noisy speech recognition has been released. The WI008 advanced front-end (ETSI 2002) utilises a two-stage Mel-warped Wiener filtering to improve the signal-to-noise ratio of a speech utterance, and then applies an SNR-dependent waveform processing to the noise-reduced signal. The resultant speech signal is further processed into MFCCs and after which a blind equalisation is applied to the cepstral coefficients. While the advanced front-end WI008 represents the state-of-art in terms of recognition accuracy for noisy speech, its main drawback is high computation load due to the use of double Wiener filtering. There is a need for an alternative front-end that is as robust as the WI008 advanced front-end but has comparable computation to that of the WI007 standard front-end.

In this work, the main focus is on feature-space compensation for a cepstral based front-end. It is demonstrated that a general framework of Mel-filterbank output compensation can be used together with cumulative distribution mapping to compensate the effects of noises. Here we extend our previous work (Choi 2004) by adding log Mel-filterbank output weighting and frame skipping to the proposed front-end processing. We also benchmark the proposed front-end against the ETSI front-ends for evaluation purpose.

The organisation of this paper is as follows. It will describe the details of the proposed front-end in Section 2. Following this in Section 3 will be some recognition experiments on the Aurora II digits database and the corresponding discussion. Finally, a summary of the conclusions will be presented in Section 4.

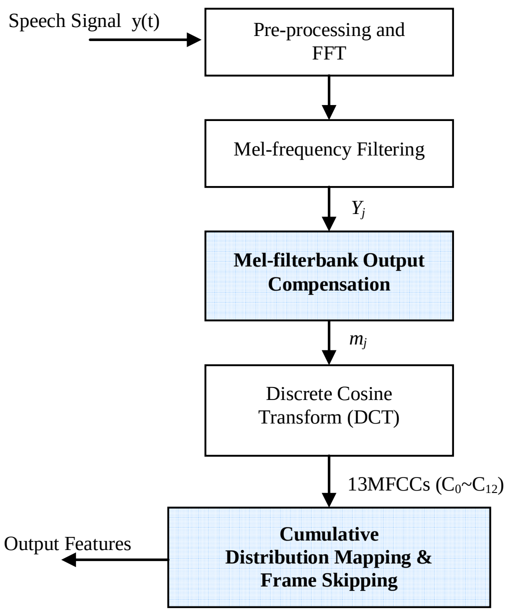
2 Front-end Processing

Typical ASR front-ends lack the ability to compensate the effects of noise on feature extraction and if a speech signal is noisy, they tend to extract more information about the noise, instead of the speech itself. Therefore a noise robust front-end needs to have knowledge about the noise and accordingly adjust the processing to extract only relevant information about the speech. To this end, we have experimented with some novel noise compensation techniques that not only account for the effects of noise but also emphasise speech information that is less susceptible to noise corruption. A high level block diagram of the proposed front-end is shown in Figure 1.

The development of the proposed front-end processing is based on the ETSI standard Mel-frequency cepstral coefficient front-end (ETSI 2000). Typically, the MFCCs (C_i) of a frame of speech data are given by:

$$C_i = \sum_{j=1}^{M} m_j Cos[\frac{i\pi}{M}(j-0.5)]; \quad m_j = \log_e(Y_j); \qquad (1)$$
$$i = 0,1,2,.....,N; N < M$$

where Y_j is the output magnitude of the j-th Mel-filterbank and M is the total number of Mel-filters in the filterbank analysis. In processing an utterance, each frame of speech data is 25ms wide and there is a 10ms time shift between current frame and the next frame of speech data (i.e. 15ms overlap between two consecutive frames).



Speech Signal y(t)
Pre-processing and FFT
Mel-frequency Filtering
Y_j
Mel-filterbank Output Compensation
m_j
Discrete Cosine Transform (DCT)
13MFCCs (C_0~C_{12})
Output Features
Cumulative Distribution Mapping & Frame Skipping

Figure 1: Block diagram of the proposed front-end (two novel processing blocks related to noise compensation are highlighted)

In this work, two more processing blocks related to noise compensation have been added to the ETSI standard MFCC front-end. The Mel-filterbank output compensation block incorporates noise spectral subtraction, spectral flooring and log Mel-filterbank output weighting into a single framework. Moreover, noise robustness is further enhanced by applying cumulative distribution mapping (CDM) with frame skipping to the resultant cepstral coefficients. A detailed description of this novel noise compensation framework is presented as in the following sub-sections.

2.1 Mel-filterbank Output Compensation

The noise robustness of the proposed front-end is enhanced by compensating the Mel-filterbank outputs according to the noise spectral characteristics. In this work, an enhanced log Mel-filterbank output is given by:

$$m_j = \alpha_j \log_e\{1+\beta_j MAX[(Y_j - \hat{N}_j), \gamma_j Y_j]\} \qquad (2)$$

where α_j, β_j, γ_j all $\in$ (0,1) are parameters to adjust the noise compensation, $\hat{N}_j$ is the noise magnitude estimate of the j-th Mel-filterbank output and MAX[.] is a function which returns the maximum value of its arguments.

Note that γ_j is used to control the degree of noise spectral subtraction (Vaseghi 2000) and β_j is used to adjust the degree of spectral flooring (Choi 2004). Here, both γ_j and β_j are assumed to be independent of the Mel-filterbank index j as we are more interested in the log Mel-filterbank output weighting and this assumption can simplify the formulation. Also these two parameters are applied globally in that they have the same values for all the speech utterances.

The motivation to incorporate log Mel-filterbank output weighting is to emphasise those filterbank outputs which are found to be more reliable and less affected by the actual noise spectral characteristics. One possible way to measure the reliability of a filterbank output is the signal-to-noise ratio (SNR). From the viewpoint of psychoacoustics (Stevens 1957), these weighing factors (α_j) are related to the spectral compression process that converts sound intensity into perceived loudness by human. So far in the literature, each of the weighting factors has been assumed to be dependent on its individual output SNR only. However, in our case, a weighting factor is also dependent on the SNRs of other filterbank outputs and it is given by:

$$\alpha_j = \frac{\log_e(1+\frac{Y_j}{\hat{N}_j})}{\sum_{k=1}^{M}\log_e(1+\frac{Y_k}{\hat{N}_k})}; \qquad \sum_{j=1}^{M}\alpha_j = 1 \tag{3}$$

The constant "1" is added to the log function to prevent it from having negative values since there may be errors in the noise estimates. In essence, α_j is basically calculated as the ratio of the SNR of a particular filterbank output to the sum of the SNRs of all the filterbank outputs. Moreover, in this case, all the weighing factors are calculated frame-by-frame dynamically based on the noise estimates from the first 10 frames of each speech utterance.

While equation (2) provides a general framework to perform the noise compensation, it is anticipated that some kind of normalisation to the dynamic ranges of the compensated cepstral coefficients would be beneficial. For this purpose, we choose to apply cumulative distribution mapping to the cepstral coefficients after noise compensation.

2.2 Cumulative Distribution Mapping

The cumulative distribution mapping (CDM) method described here is based on the use of histogram equalisation (HE) originally developed for image processing (Russ 1995). The use of the HE method for noise compensation in front-end processing of speech can also be found in (Dharanipragada and Padmanabhan 2000). The main idea of this method is to map the distribution of a time sequence of noisy speech features into a target distribution with a pre-defined probability density function (PDF). In our case, it is assumed that for a given feature value v_o, the mapping relationship would be:

$$\int_{v=-\infty}^{v_o} f(v)dv = \int_{z=-\infty}^{z_o} h(z)dz \text{ ; or } F_v(v_o) = F_z(z_o) \tag{4}$$

where $F_v(v)$ is the corresponding cumulative distribution function (CDF) of a given set of noisy speech features and $F_z(z)$ is the target CDF, $f(v)$ and $h(z)$ are the respective PDFs. From equation (4), we have

$$z_o = F_z^{-1}[F_v(v_o)] \tag{5}$$

Therefore the required mapping from a given speech feature v_o into the corresponding target feature z_o is represented by equation (5). Typically $h(z)$ is assumed to be a Gaussian as in the literature of histogram equalisation.

On the other hand, there is no particular strong reason, other than easier implementation, that one has to assume $h(z)$ to be Gaussian. In fact, we have observed that the left tail region of the distribution of a normalised feature may not be that useful as it represents mainly the range of more noisy features. Based on this observation, we have developed the novel use of a truncated Gaussian as target distribution. Mathematically, the additional constraint is given by:

$$z_o = \begin{cases} F_z^{-1}[F_v(v_o)], & \text{if } F_v(v_o) \ge \theta_{th} \\ SKIP, & \text{otherwise} \end{cases}; \qquad 0 \le \theta_{th} < 1 \tag{6}$$

where θ_{th} is a constant that determines the fraction of features to be discarded, and "SKIP" denotes a function that skips the current frame of speech data and does not output any feature value. In the current implementation, we perform the skipping of a whole feature vector based only on C_0 (zeroth-order cepstral coefficient) as it indicates the energy level of a frame of speech data. Moreover, the $h(z)$ is assumed to be a Gaussian with zero mean and unity variance. In the experiments, CDM is applied only to the static feature vector which consists of 13 MFCCs (C_0 ~ C_{12}) and each cepstral coefficient is normalised individually.

3 Experimental Results

The proposed front-end has been evaluated on the Aurora II database (Hirsch and Pearce 2000). This database contains noisy connected digits, which were created by adding various types of noises at different SNRs to the original clean utterances (i.e. utterances with high SNRs). There are three test sets in the database and they contain 8 types of additive noises. Each of the test sets *A* and *B* contains about 28K utterances and the test set *C* is about half that size. The test set *C* includes channel distortion as well. The SNRs of the test data range from -5 dB to more than 20 dB. The training data consist of another 8440 clean utterances.

3.1 Experimental Setup

All the pre-processing and Mel filtering of a speech signal in the proposed front-end followed the ETSI standard MFCC front-end. The static feature vector of our front-end consisted of 13 MFCCs (C_0 ~ C_{12}). This static feature vector was appended with their corresponding 1st-order and 2nd-order time derivatives to form a resultant vector with 39 coefficients for speech recognition at the backend, as per the Aurora evaluation framework. Hidden Markov modelling (HMM) (Rabiner and Juang 1993) was used for the speech recognition experiments. Each model was represented by a continuous density HMM with left-to-right configuration. Digit models had 16 states with 3 Gaussians per state, while the silence model had 3 states with 6 Gaussians per state. An inter-digit silence model with 1 state was also used, and it was tied with the middle state of the silence model.

3.2 Comparison of Accuracy and Robustness

We followed the official Aurora evaluation framework in that average recognition accuracy for each test set is calculated from the recognition results for those test data with SNRs from 0 dB to 20dB only. In all the experiments reported here, the spectral flooring parameter β_j and the spectral subtraction parameter γ_j for the proposed front-end were set to 0.001 and 0.4 respectively, as determined empirically in some preliminary experiments. Note that the 1st-order and the 2nd-order time derivatives of a static feature vector were generated after the static features had been compensated and normalised.

The first set of experiments investigated the effect of the frame skipping threshold (θ_{th}) on the recognition accuracy of the proposed front-end. The experimental results obtained with various values of the threshold are summarised as shown in Table 1.

Table 1: Average digit accuracies (%) for Aurora test sets, proposed front-end with various thresholds (θ_{th}) for skipping frames

θ_{th}	Test *A*	Test *B*	Test *C*	Avg.
0.00#	83.65	84.00	82.74	83.46
0.03	84.47	84.90	83.58	84.32
0.05	84.57	84.93	83.76	84.42
0.06	84.71	85.22	83.91	84.61
0.07	84.98	85.41	84.08	84.82
0.08	85.06	85.49	84.21	84.92
0.09	85.10	85.50	84.10	84.90
0.10	85.08	85.61	83.95	84.88

# No frame skipping in this case

As observed from Table 1, the incorporation of frame skipping in CDM does improve the accuracy of the proposed front-end and the optimal threshold for achieving the best average accuracy is found to be 0.08. Since the frame skipping is applied to feature vectors with smaller value of C_0, this is equivalent to removing speech segments which have lower frame energy. Obviously, these segments are less reliable in discriminating between different speech sounds, as they can potentially contain more information about the noise than the speech signal itself.

Furthermore, it may be observed from the table that the optimal frame skipping threshold is different for different test sets. It seems that some kind adaptive threshold according to noise condition and characteristic would be beneficial. Nevertheless, for the Aurora digit strings, skipping about 10% of the feature vectors in an utterance is seemed to be reasonable.

Table 2: Average digit accuracies (%) for Aurora test sets, comparing proposed front-end (θ_{th}=0.08) with ETSI MFCC front-ends

Front-end	Test *A*	Test *B*	Test *C*	Avg.	% Improv*
ETSI std.	61.34	55.75	66.14	61.08	0.0
ETSI adv.	86.20	85.24	84.72	85.39	62.5
Proposed	85.06	85.49	84.21	84.92	61.3

*% Improvement is measured in terms of relative error rate reduction reference to the ETSI standard front-end

The performances of the proposed front-end with θ_{th}=0.08 were compared with those of the ETSI MFCC front-ends and the results are shown in Table 2. From the table, it can be observed that the proposed front-end performs much better than the ETSI standard MFCC front-end in terms of average recognition accuracy, while it achieves comparable recognition accuracy with the ETSI advanced front-end. Although for the test set *B*, the proposed front-end seems to perform marginally better than the advanced front-end (85.49% vs. 85.24%), the difference in accuracy is found to be not statistically significant. The fact that both the proposed front-end and the advanced front-end have similar accuracy is noteworthy since the proposed front-end requires only about half the computation load of the advanced front-end, as it will be shown later in Section 3.3.

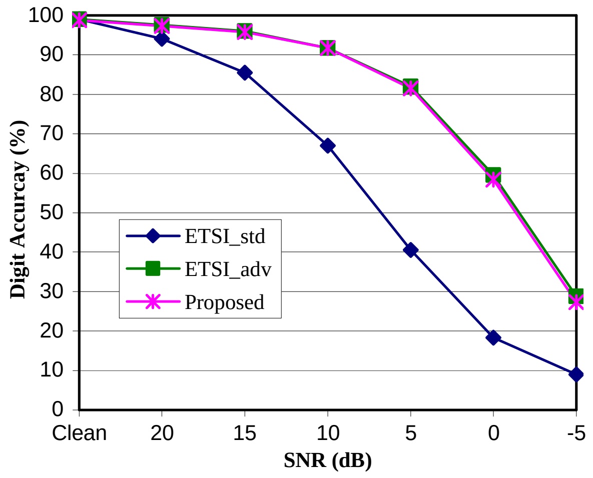


Figure 2: Average recognition results for Aurora test sets, proposed front-end (θ_{th}=0.08) compared with ETSI MFCC front-ends by SNR

To get an insight on how the proposed front-end is performing at different noise levels, a break-down of the recognition results according to individual SNRs and averaged across all three test sets is shown in Figure 2. Also shown in the figure are those corresponding results for the ETSI MFCC front-ends.

As observed from Figure 2, both the proposed front-end and the advanced front-end perform similarly at different SNRs. On the other hand, both the proposed front-end and the advanced front-end perform much better than the ETSI standard MFCC front-end, particularly in the noisier conditions. In some cases, more than double of the recognition accuracy can be achieved by using the proposed front-end (e.g. at 5dB SNR).

To illustrate the performances of the front-ends for different noise types, the average recognition accuracy over 0 to 20 dB SNRs obtained by each front-end for each type of noisy speech data in test set *A* is plotted in Figure 3. From the figure, it can be observed that the proposed front-end achieves higher digit accuracy than the ETSI advanced front-end for the babble-type noisy speech (other people talking at background causing the noises). The difference in accuracy (84.74% vs. 82.21%) is found to be statistically significant (z=6.195, p<0.001, two tailed). Overall, the advanced front-end is found to achieve marginally better accuracy than the proposed front-end for the other types of noisy speech.

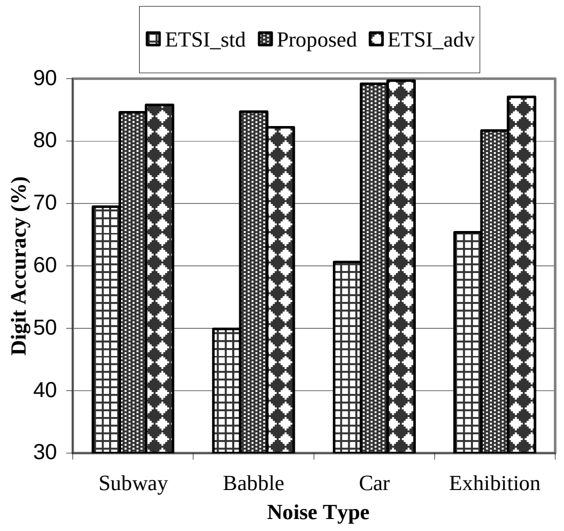


Figure 3: Recognition results for Aurora test set *A*, proposed front-end (θ_{th}=0.08) compared with ETSI MFCC front-ends by noise type

Similarly, the recognition results by noise type for test sets *B* and *C* are also shown in Figure 4. Note that the (C) following the name of a noise type in the figure denotes speech data from test set *C* which also contains additional channel distortion.

Again it can be observed from Figure 4 that the proposed front-end performs as good as the advanced front-end for all the noise types and the proposed front-end achieves a better accuracy for the restaurant-type noisy speech. This better accuracy (82.52% vs. 81.11%) is found to be statistically significant (z=3.324, p<0.001, two tailed). It seems that the proposed front-end is particularly effective in handling background noises due to other people talking at the same time.

Overall the previous two figures demonstrate that the proposed front-end is much more consistent and robust than the ETSI standard MFCC front-end in recognising different types of noisy speech, and it is as noise robust as the ETSI advanced front-end in most of the cases.

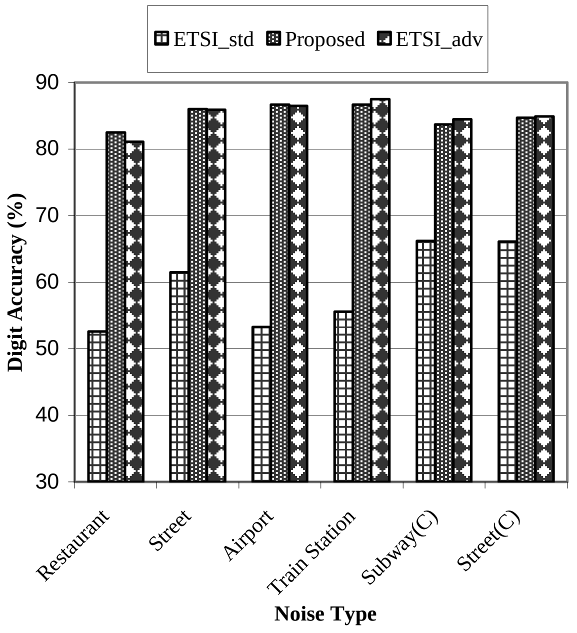


Figure 4: Recognition results for Aurora test sets *B*, and *C*, proposed front-end (θ_{th}=0.08) compared with ETSI MFCC front-ends by noise type

3.3 Comparison of Computation Load

In order to estimate the computational complexity of the proposed front-end processing, the ETSI standard, the ETSI advanced and the proposed front-end were run on the Aurora II multi-condition training data, and the duration was recorded as shown in Table 3. No other processes were running on the processor at the time. The multi-condition training set contains utterances with 4 different noise types (subway, babble, car and exhibition) and 5 SNRs (5 to 20dB and "clean"). In total, there are 8440 utterances in the training set (422 utterances per condition).

Table 3: Comparison of running times on a 2.66 GHz processor with 2 GB RAM for front-end processing of Aurora multi-condition training set (8440 utterances)

Front-end	ETSI std.	Proposed	ETSI adv.
Time (s)	132	158	325

On average, the computation load of the proposed front-end was found to be about 20% more than that of the ETSI standard MFCC front-end, but only about 49% that of the ETSI advanced front-end. It took an average of about 19ms for the proposed front-end to process an utterance. The higher computational load of the ETSI advanced front-end is expected, as the advanced front-end applies Wiener filtering twice to a speech signal based on time-domain convolution.

Compared with the ETSI advanced front-end, the much lighter computation requirement of the proposed front-end can be a distinguished advantage for applications running on handheld devices. Moreover, the proposed front-end is easier to be implemented on fixed-point processors used by most handheld devices.

4 Conclusions

A new and noise robust front-end based on the combined application of Mel-filterbank output compensation and cumulative distribution mapping with frame skipping has been proposed. Experimental results on the Aurora II speech database have revealed the effectiveness of the novel combination of these noise compensation methods. The proposed front-end achieves an average digit accuracy of 84.92% for the three test sets with clean HMM training. Compared with the ETSI standard Mel-cepstral front-end, the proposed front-end has been able to provide a relative error rate reduction of more than 61%. Moreover, the proposed front-end can provide comparable recognition accuracy with the ETSI advanced front-end, at less than half the computation load. Possible future extension work includes the use of dynamic noise estimates to handle non-stationary noises, the replacement of the simple spectral flooring with a more advanced temporal masking algorithm and the use of adaptive threshold for frame skipping.

5 References

Choi, E. (2004): Noise Robust Front-end for ASR using Spectral Subtraction, Spectral Flooring and Cumulative Distribution Mapping. *Proc. 10th Australian Int. Conf. on Speech Science and Technology*, pp. 451-456.

Deng, Li. and Huang, X. (2004): Challenges in Adopting Speech Recognition. *Communications of the ACM*, Vol. 47, No.1, pp. 69-75.

Dharanipragada, S. and Padmanabhan, M. (2000): A Nonlinear Unsupervised Adaptation Technique for Speech Recognition. *Proc. Int. Conf. on Spoken Language Processing*, Vol. 4, pp. 556-559.

Ephraim, Y. (1992): A Bayesian Estimation Approach for Speech Enhancement Using Hidden Markov Models. *IEEE Trans. Signal Processing*, Vol. 40, No. 4, pp. 725-735.

ETSI (2000): Speech Processing, Transmission and Quality Aspects (STQ); Distributed Speech Recognition; Front-end Feature Extraction Algorithm; Compression Algorithms. *ETSI standard document ES 201 108*.

ETSI (2002): Speech Processing, Transmission and Quality Aspects (STQ); Distributed Speech Recognition; Advanced Front-end Feature Extraction Algorithm; Compression Algorithm. *ETSI standard document ES 202 050*.

Hermansky, H. (1990): Perceptual Linear Predictive (PLP) Analysis of Speech. *Journal Acoustical Society of America* (*JASA)*, Vol. 87 (4), pp. 1738-1752.

Hirsch, H.G. and Pearce, D. (2000): The AURORA Experimental Framework for the Performance Evaluation of Speech Recognition Systems Under Noise Conditions. *Proc. ISCA ITRW ASR2000*, pp. 181-188.

Huang, C., Wang, H. and Lee, C. (2001): An SNR-Incremental Stochastic Matching Algorithm for Noisy Speech Recognition. *IEEE Trans. Speech and Audio Processing*, Vol. 9, No. 8, pp. 866-873.

Rabiner, L.R. and Juang, B.H. (1993): *Fundamentals of Speech Recognition*. Prentice Hall, Englewood Cliffs, New Jersey.

Russ, J.C. (1995); *The Image Processing Handbook*. 2nd ed., CRC Press.

Sankar, A. and Lee, C.H. (1996): A Maximum Likelihood Approach to Stochastic Matching for Robust Speech Recognition. *IEEE Trans. Speech and Audio Processing*, Vol. 4, pp. 190–202.

Stevens, S.S. (1957): On the Psychological Law. *Psychological Review*, Vol. 64, pp. 153-181.

Vaseghi, S.V. (2000): *Advanced Digital Signal Processing and Noise Reduction*. Wiley Press.

Yao, K., Paliwal, K.K. and Nakamura, S. (2001): Sequential Noise Compensation by a Sequential Kullback Proximal Algorithm. *Proc. European Conf. on Speech Communication and Technology*, pp. 1139-1142.

Zhang, Z. and Furui, S. (2004): Piecewise-linear Transformation-based HMM Adaptation for Noisy Speech. *Speech Communication*, Vol. 42, Issue 1, pp. 43-58.

Segregated Failures Model for Availability Evaluation of Fault-Tolerant Systems

Sergiy A. Vilkomir[1] **David L. Parnas**[1] **Veena B. Mendiratta**[2]
Eamonn Murphy[3]

[1]Software Quality Research Laboratory (SQRL), Department of Computer Science and Information Systems, University of Limerick, Limerick, Ireland,
Email: Sergiy.Vilkomir@ul.ie David.Parnas@ul.ie
[2]Bell Laboratories, Lucent Technologies, Naperville, IL, USA,
Email: Veena@lucent.com
[3]Department of Mathematics and Statistics, University of Limerick, Limerick, Ireland,
Email: Eamonn.Murphy@ul.ie

Abstract

This paper presents a method of estimating the availability of fault-tolerant computer systems with several recovery procedures. A segregated failures model has been proposed recently for this purpose. This paper provides further analysis and extension of this model. The segregated failures model is compared with a Markov chain model and is extended for the situation when the coverage factor is unknown and failure escalation rates must be used instead. This situation is illustrated in detail by estimating availability of a Lucent Technologies Reliable Clustered Computing architecture. For this example, numeric values are provided for availability indexes and the contribution of each recovery procedure to total system availability is analysed.
Keywords: software, fault-tolerance, availability, reliability, recovery, failures model.

1 Introduction

Some computer systems allow a variety of ways to restore service after a failure. We say such systems have several recovery procedures. For example, recovery procedure 1 can be a restart of a current application. If the restart of the application does not succeed, the computer can be restarted (recovery procedure 2). If the computer is still down after the restart, a repair or replacement is necessary (recovery procedure 3).

An example of an industrial computer system with several recovery procedures is a Lucent Technologies Reliable Clustered Computing (RCC) application (Hughes-Fenchel 1997). One of the basic recovery procedures for RCC is a switchover to a spare computer. Reliability and availability of systems with several recovery procedures have been studied by Hoeflin & Mendiratta (1995), Lyu & Mendiratta (1999), and Mendiratta (1998) for RCC products and by Ibe, Howe & Trivedi (1989), Sun, Han & Levendel (2001), and Sun, Han & Levendel (2003) for other systems.

Markov chains have been used as the most popular approach to reliability and availability investigation of systems with several recovery procedures (Ibe, Howe & Trivedi 1989, Lyu & Mendiratta 1999, Mendiratta 1998, Sun, Han & Levendel 2003). In Vilkomir *et al.* (2005), we have suggested a simpler analytical method of availability evaluation as an alternative. This method allows calculation of availability without using special tools and yields an assessment of the impact of every recovery procedure to system availability.

This paper continues the investigation started in Vilkomir *et al.* (2005) and considers the further analysis and extension of a segregated failures model. The paper is structured as follows. Section 2 presents a brief review of the segregated failures model based on Vilkomir *et al.* (2005) and then continues with new analysis and an extension of this model. In section 3, we compare our model with the Markov chain model. Small examples show the difference of approaches to availability evaluation for these two models and the advantages of using the segregated failures model are presented. We extend the model for the situation when coverage factors are unknown and show how to use rates of escalation instead. In section 4, a case study of an RCC application is considered in order to illustrate the extension of the segregated failures model. Because the RCC application has been previously considered by Lyu & Mendiratta (1999) using the Markov chain model, it provides an additional chance to compare the two approaches.

2 Segregated failures model of a system with several recovery procedures

Consider a system with n ($n > 1$) different recovery procedures. For every procedure from 1 to $n-1$, the result of the recovery can be either successful or unsuccessful. Level n recovery is always successful. When a failure occurs, recovery procedures are applied sequentially starting from level 1. If the recovery procedure at level 1 is unsuccessful, the level 2 procedure is applied, etc. However, if at any level it is determined for a specific failure that the usage of next recovery levels will not help, these levels can be skipped and the procedure of the last level n can be applied directly. Thus, there are three possibilities when a failure recovery is attempted at level $i, 1 \leq i < n$:

- The recovery is successful.
- The recovery is unsuccessful and the next level procedure will be applied (the failure is *escalated* from level i to the next level $i+1$).
- The recovery is unsuccessful and the highest level procedure will be applied (the failure is escalated to level n).

These possibilities reflect real Lucent Technologies Reliable Clustered Computing applications as considered below in the Case Study section. The model can be extended to consider additional hypothetic possibilities such as:

- Skipping some restoration levels but not all of them.
- Using diagnostics that allows changing the order of recovery procedures for every specific failure depending on the nature of the failure.

We do not address these extensions in this paper but they are straightforward.

The ability of the recovery procedure to successfully restore a failure is described by a *coverage factor*. More precisely, a coverage factor $p_{rec,i}$ of the recovery level i is a conditional probability that a failure is successfully recovered at level i given that this failure is served at level i. In that way, a coverage factor $p_{rec,i}$ is the probability of the first of three mentioned above possibilities. Denote as $p_{next,i}$ the probability of second and as $p_{last,i}$ the probability of the third possibility.

When a recovery procedure is successful, we assume it provides full (not partial) recovery. This assumption is valid for many practical situations including the case study in section 4. Another assumption is that recovery duration is a random variable. This assumption is a traditional one and does not require special explanations. We consider the same distribution of recovery time whether the procedure is successful or not. To model it, we use restoration rate μ_i or mean restoration time $\tau_i = 1/\mu_i$. These indexes describe here only the duration of restoration, not its result (successful or unsuccessful). To highlight it, we will also use a term *mean processing time* for τ_i.

The main idea of the proposed approach is classifying processor failures into several types and evaluating the influence of each type of failure on the availability of the whole system. We propose the following definition of failure of *type i*: a failure f is said to be a failure of *type i* if and only if i is the lowest level where this failure is successfully recovered.

The described division of failures into types and the main parameters of the model are shown in Fig. 1, where λ is a system failure rate.

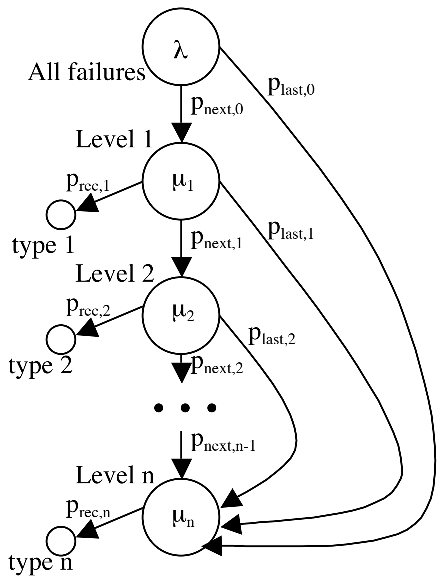


Figure 1: Segregated failures model

In Vilkomir *et al.* (2005), we had proposed using six main steps to evaluate availability according the segregated failures model.

At step 1, different types of failures should be determined corresponding to the recovery levels.

At step 2, probability p_{type_k} that a failure belongs type k is evaluated for each type k. For type 1,

$$p_{type_1} = p_{rec,1} \times p_{next,0} \tag{1}$$

For types k, $1 < k < n$,

$$p_{type_k} = p_{rec,k} \times \prod_{i=1}^{k-1} p_{next,i} \times p_{next,0} \tag{2}$$

For type n,

$$p_{type_n} = (p_{last,1} + \sum_{j=2}^{n-2} (p_{last,j} \times \prod_{i=1}^{j-1} p_{next,i}) + \\ + \prod_{i=1}^{n-1} p_{next,i}) \times p_{next,0} + p_{last,0} \tag{3}$$

At step 3, the failure rate λ_{type_k} is evaluated for failures of each type k:

$$\lambda_{type_k} = \lambda \times p_{type_k} \tag{4}$$

where p_{type_k} are determined by (1), (2), and (3) .

At step 4, the restoration rate μ_{type_k} is evaluated for failures of each type k. For failures of type k, the mean restoration time τ_{type_k} includes mean processing time τ_k at level k and also time which has been unsuccessfully spent on recovery at the previous levels:

$$\tau_{type_k} = \sum_{i=1}^{k} \tau_i \tag{5}$$

The restoration rate is:

$$\mu_{type_k} = \frac{1}{\sum_{i=1}^{k} \frac{1}{\mu_i}} \tag{6}$$

where μ_i is restoration (service) rate for the recovery procedure at level i.

At step 5, the availability is evaluated for failures of each type k. The expected down time T_{dk} during a fixed period of time T relative to failures of type k is:

$$T_{dk}(T) = T(1 - A_k) = T \frac{\lambda_{type_k}}{\lambda_{type_k} + \mu_{type_k}} \tag{7}$$

where $A_k = \mu_{type_k}/(\lambda_{type_k} + \mu_{type_k})$ - availability factor.

When $\lambda_{type_k} \ll \mu_{type_k}$, it is possible to use the approximate value of the down time per year:

$$T_{dk} = \lambda_{type_k} \tau_{type_k} \tag{8}$$

calculating λ_{type_k} in 'failures per year'and τ_{type_k} in minutes.

At step 6, the down time of the whole system is evaluated:

$$T_d = \sum_{k=1}^{n} T_{dk} \tag{9}$$

3 Analysis and further extension of the model

3.1 Comparison with a Markov chain model

In this section we illustrate similarities and differences between the Markov chain model and the segregated failures model. The Markov chain model for a system with several recovery levels is illustrated in Fig. 2. Despite a certain similarity between Fig. 1 (Segregated failures model) and Fig. 2 (Markov chain model), they have different meanings.

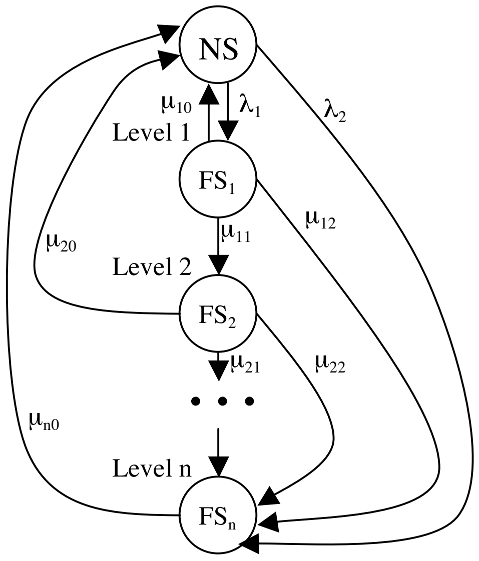


Figure 2: Markov chain model.

The circles in Fig. 2 represent states of a system: one working state and several faulted states for every recovery level. The arrows represent transitions between system states. In contrast, the circles in Fig. 1 represent sets of failures. The arrows represent relationships between these sets, i.e., how one set of failures is divided into other sets (with corresponding probabilities). The two models use the same input data and lead to the same results but use different approaches to system availability evaluation.

The Markov chain model is a powerful mathematical approach and allows modelling of many aspects of a system's behaviour, not just availability. Using the Markov chain model, the probabilities of system states are evaluated. Knowing the probability of a normal (working) state, the system availability can be evaluated. However, calculations based on this model can be quite complicated (solving the Chapman-Kolmogorov equations). Special tools are often required for this type of analysis. The segregated failures model is designed for a specific purpose - availability evaluation of systems with several recovery procedures. System states are not considered and an impact of different types of failures on system availability is considered instead. In contrast to the Markov chain model, calculations are very simple and do not required of using any tools.

The segregated failures model is proposed not instead of but in addition to the Markov chain model. Taking into account the computational complexity of the Markov chain model, we believe it is useful to have a simple engineering analytical method of availability evaluation. To illustrate this, consider an application of both models to the following simple toy example:

- A system has two different recovery procedures.
- The probability that a failure is recovered by the first procedure is 2/3.
- The mean restoration time for the first recovery procedure is 30 times less then for the second one.

Both models for this case are represented in Fig. 3, where

- λ - system failure rate.
- μ - recovery rate for the second procedure.
- $P_i, i = 0, 1, 2$ - probabilities of system states.

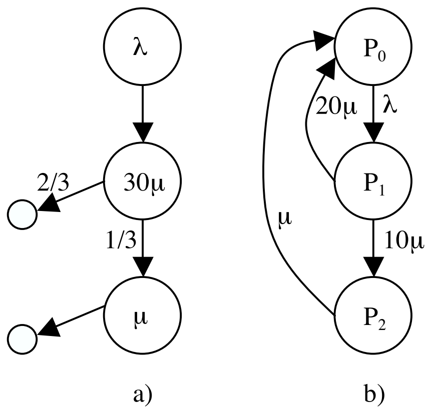


Figure 3: System with two recovery procedures: a) Segregated failures model and b) Markov chain model

Application of the segregated failures model for this example is so simple that it requires only mental calculations. Thus, the failure rates for failures of the each type from (4) are

$$\lambda_{type_1} = \frac{2}{3}\lambda; \quad \lambda_{type_2} = \frac{1}{3}\lambda \tag{10}$$

The mean restoration times for failures of the each type from (5) are

$$\tau_{type_1} = \frac{1}{30\mu}; \quad \tau_{type_2} = \frac{1}{30\mu} + \frac{1}{\mu} = \frac{31}{30\mu} \tag{11}$$

Finally, the system down time can be found using (9):

$$T_d = \frac{2}{3}\lambda\frac{1}{30\mu} + \frac{1}{3}\lambda\frac{31}{30\mu} = \frac{11\lambda}{30\mu} \tag{12}$$

Application of the Markov chain model is slightly more complicated. We need to solve the following simultaneous equations:

$$\lambda P_0 = 20\mu P_1 + \mu P_2 \tag{13}$$

$$30\mu P_1 = \lambda P_0 \tag{14}$$

$$\mu P_2 = 10\mu P_1 \tag{15}$$

$$P_0 + P_1 + P_2 = 1 \tag{16}$$

Transposing (14) for P_1 and (15) for P_2 and substituting P_1 and P_2 into (16) gives

$$P_0 + \frac{\lambda}{30\mu}P_0 + \frac{\lambda}{3\mu}P_0 = 1 \tag{17}$$

and

$$P_0 = \frac{30\mu}{11\lambda + 30\mu} \tag{18}$$

The system down time during time period T can be expressed as

$$T_d = (1 - P_0)T \tag{19}$$

Considering down time during $T = 1year$ and using (18) , we finally have

$$T_d = \frac{11\lambda}{11\lambda + 30\mu} \tag{20}$$

In practice usually $\lambda_{type_k} \ll \mu_{type_k}$ and from (20) the approximate value of the down time is

$$T_d = \frac{11\lambda}{30\mu} \tag{21}$$

which completely coincides with result (12) of the segregated failures model.

The following conclusions can be drawn from the example:

- The segregated failures model provides the approximate values of the down time which are very close to the accurate values. Thus, if $\lambda = 10$ *per year* and $\mu = 1$ *per hour*, the accurate value of T_d according (20) is 219.91 *min per year*. The approximate value of T_d according to (12) or (21) is 220.00 *min per year*. The difference is only 0.04% and is negligible, especially taking into account an approximation of the input data.
- The complexity of calculations according to the Markov chain model increases when the number of recovery procedures increases. At the same time, the complexity of calculations according to the segregated failures model changes insignificantly. Thus, the benefit of using the segregated failures model increases when more recovery procedure are used.
- Both the Markov chain model and the segregated failures model allow us to evaluate the system down time. However, the segregated failures model also provides a separate evaluation of the down times for each recovery procedure. This in turn allows us to analyze availability in more detail and to find ways to improve availability.

3.2 Rates of escalation instead of the coverage factor

In this section we propose an extension of the segregated failures model for the situation when the input data are different from what was considered previously. As was mentioned in Section 2, we assumed that some conditional (given that level i procedure is applied) probabilities are known. Specifically, probability that a failure recovery is successful ($p_{rec,i}$) or that a failure recovery is unsuccessful and that the next recovery level is either level $i+1$ ($p_{next,i}$) or last level n ($p_{last,i}$). For applications of the Markov chain model, explicit rates of transitions between system states are often used as input data instead of these probabilities. This situation is also possible for the segregated failures model. In this case, the input data for the model are:

- $\mu_{rec,i}$ - successful failure recovery rate.
- $\mu_{next,i}$ - rate of the escalation from level i to the next level $i+1$.
- $\mu_{last,i}$ - rate of the escalation from level i to the last level n.

These three possibilities are mutually exclusive and exhaustive. So the failure *processing* rate μ_i at level i (i.e. failure exit rate regardless of the results of recovery) is a sum of rates for these possibilities:

$$\mu_i = \mu_{rec,i} + \mu_{next,i} + \mu_{last,i} \tag{22}$$

The best way to evaluate availability in this situation is to express $p_{rec,i}$, $p_{next,i}$, and $p_{last,i}$ using $\mu_{rec,i}$, $\mu_{next,i}$ and $\mu_{last,i}$ and then to apply the basic segregated failures model described in Section 2. Each probability is determined as a ratio of the corresponding rate to the failure processing rate of the whole procedure:

$$p_{rec,i} = \frac{\mu_{rec,i}}{\mu_i} = \frac{\mu_{rec,i}}{\mu_{rec,i} + \mu_{next,i} + \mu_{last,i}} \tag{23}$$

$$p_{next,i} = \frac{\mu_{next,i}}{\mu_i} = \frac{\mu_{next,i}}{\mu_{rec,i} + \mu_{next,i} + \mu_{last,i}} \tag{24}$$

$$p_{last,i} = \frac{\mu_{last,i}}{\mu_i} = \frac{\mu_{last,i}}{\mu_{rec,i} + \mu_{next,i} + \mu_{last,i}} \tag{25}$$

To apply the model from Section 2, we also need to express mean *service* (processing) time τ_i at level i. For traditional systems with one recovery procedure, the mean restoration time is a reciprocal value of the failure restoration rate. For systems with several recovery procedures, there are different rates for each level, in particular, rates of successful failure recovery $\mu_{rec,i}$ and failure processing rate μ_i. Because we assume that the mean processing time for a specific level is the same for all failures and independent of restoration results, this time is a reciprocal value of the failure processing rate, not of the successful recovery rate. In other words,

$$\tau_i = \frac{1}{\mu_i} = \frac{1}{\mu_{rec,i} + \mu_{next,i} + \mu_{last,i}} \tag{26}$$

Using (23) - (26) allows us to evaluate availability of a system with known explicit rates of escalation, leading to the situation described in Section 2.

4 A Case Study: Describing recovery policy by rates of escalation without the use of coverage factor

4.1 A model

As an example of a model with rates of escalation we consider a hypothetical RCC application, which has been analyzed by Lyu & Mendiratta (1999) using a Markov chain model. We reuse notation and inputs from Lyu & Mendiratta (1999) in our model which allows us to compare different approaches to availability evaluation.

The model has four recovery procedures:

- Fault Detection and Recovery. A small number of hardware and software faults are detected by the watchdog and recovery is fully automatic. The internal data are not saved and the application is restarted at the initial internal state.
- Volatile Data Recovery. Periodic checkpointing is used and the critical volatile data are saved. The process automatically restarts at the most recent checkpointed internal state.

- Persistent Data Recovery. Replication of the persistent data on a backup disk is carried out. This ensures data consistency when the application is automatically recovered on the backup node.
- Manual Repair. For all hardware and software faults when attempts of automatic recovery are not successful, manual intervention is used. In addition, a small set of faults is detected for manual repair before applying procedures of the automatic recovery.

A diagrammatic representation of the model is shown in Fig. 4.

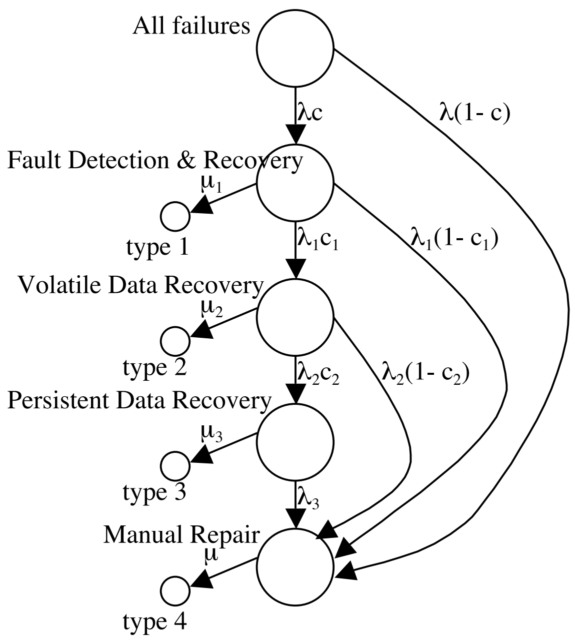


Figure 4: Segregated failures model with rates of escalation.

Model inputs are the following:

- λ - total failure rate.
- $\lambda_i, i = 1, 2, 3$ - level i to $i + 1$ escalation rate.
- $\mu_i, i = 1, 2, 3$ - recovery rate at level i.
- μ - manual repair rate.
- c - fault detection coverage.
- $c_i, i = 1, 2$ - level i to $i + 1$ coverage.

The values of input data are the following (Lyu & Mendiratta 1999):
λ = 10, 20, 30 failures per year
λ_1 = 30 exits per hour
λ_2 = 1800 exits per hour
λ_3 = 100 exits per hour
μ_1 = 30, 60 recoveries per hour
μ_2 = 1800, 3600 recoveries per hour
μ_3 = 3600 recoveries per hour
μ = 0.25 repair per hour
c, c_1, c_2 = 0.9, 0.99

4.2 Availability evaluation

Step 1: The model has the following four failure types:

- Type 1: failures restored by the fault detection and recovery procedure.
- Type 2: failures restored by the volatile data recovery procedure.
- Type 3: failures restored by the persistent data recovery procedure.
- Type 4: failures restored by the manual repair procedure.

Step 2: To turn from rates of escalation to coverage factors, let us calculate conditional probabilities $p_{rec,i}$, $p_{next,i}$, and $p_{last,i}$. The application of (23) - (25) gives the following:

$$p_{rec,i} = \frac{\mu_i}{\mu_i + \lambda_i}, i = 1, 2, 3 \quad (27)$$

$$p_{next,i} = \frac{\lambda_i c_i}{\mu_i + \lambda_i}, i = 1, 2 \quad (28)$$

$$p_{next,3} = \frac{\lambda_3}{\mu_3 + \lambda_3} \quad (29)$$

$$p_{last,i} = \frac{\lambda_i(1 - c_i)}{\mu_i + \lambda_i}, i = 1, 2 \quad (30)$$

Now we can calculate p_{type_i} applying (1) - (3):

$$p_{type_1} = \frac{c\mu_1}{\mu_1 + \lambda_1} \quad (31)$$

$$p_{type_2} = \frac{c\lambda_1 c_1 \mu_2}{(\mu_1 + \lambda_1)(\mu_2 + \lambda_2)} \quad (32)$$

$$p_{type_3} = \frac{c\lambda_1 c_1 \lambda_2 c_2 \mu_3}{(\mu_1 + \lambda_1)(\mu_2 + \lambda_2)(\mu_3 + \lambda_3)} \quad (33)$$

$$p_{type_4} = c \times (\frac{\lambda_1(1 - c_1)}{(\mu_1 + \lambda_1)} + \frac{\lambda_1 c_1 \lambda_2(1 - c_2)}{(\mu_1 + \lambda_1)(\mu_2 + \lambda_2)} + \\ + \frac{\lambda_1 c_1 \lambda_2 c_2 \lambda_3}{(\mu_1 + \lambda_1)(\mu_2 + \lambda_2)(\mu_3 + \lambda_3)}) + 1 - c \quad (34)$$

Step 3: use (4) and values of $p_{rec,i}$, $p_{next,i}$, and $p_{last,i}$ (obtained at Step 2) for the calculation of λ_{type_i}.

Step 4: For the evaluation of the mean restoration time τ_{type_i} for failures of the every type i, we firstly calculate the mean processing time τ_i for the every level i using formulas $\tau_i = \frac{1}{\mu_i + \lambda_i}, i = 1, 2, 3$ and $\tau_4 = \frac{1}{\mu_4}$. Then we find τ_{type_i} using (5). The results of the calculation are shown in Table 1.

Level/ Type	Mean restoration time	$\mu_1 = 30$ $\mu_2 = 1800$	$\mu_1 = 60$ $\mu_2 = 3600$
1	τ_1	1.0	0.67
	τ_{type_1}	1.0	0.67
2	τ_2	0.02	0.01
	τ_{type_2}	1.02	0.68
3	τ_3	0.02	0.02
	τ_{type_3}	1.03	0.7
4	τ_4	240	240
	τ_{type_4}	241.03	240.7

Table 1: Mean restoration time (minutes).

Step 5 - 6: The intermediate results and values T_{di} and T_d of the down time according to (8) - (9) are presented in Table 2 for $\mu_1 = 30, \mu_2 = 1800$ and Table 3 for $\mu_1 = 60, \mu_2 = 3600$.

The comparison of these results with the results from Lyu & Mendiratta (1999), where the same case study has been analyzed using the Markov chain model, shows that the values of the down time from

Type of failures	Failure rate and down time	$c_i = 0.99$			$c_i = 0.9$		
		$\lambda = 10$	$\lambda = 20$	$\lambda = 30$	$\lambda = 10$	$\lambda = 20$	$\lambda = 30$
1	λ_{type_1}	4.95	9.9	14.85	4.50	9.00	13.50
	T_{d1}	5.0	9.9	14.9	4.5	9.0	13.5
2	λ_{type_2}	2.45	4.9	7.35	2.03	4.05	6.07
	T_{d2}	2.5	5.0	7.5	2.1	4.1	6.2
3	λ_{type_3}	2.36	4.72	7.08	1.77	3.55	5.32
	T_{d3}	2.4	4.9	7.3	1.8	3.7	5.5
4	λ_{type_4}	0.24	0.48	0.72	1.7	3.40	5.11
	T_{d4}	57.9	115.7	173.6	410.2	820.5	1230.7
System down time T_d		68	135	203	419	837	1256

Table 2: Failure rates (per year) and expected down time (minutes per year) for $\mu_1 = 30, \mu_2 = 1800$.

Type of failures	Failure rate and down time	$c_i = 0.99$			$c_i = 0.9$		
		$\lambda = 10$	$\lambda = 20$	$\lambda = 30$	$\lambda = 10$	$\lambda = 20$	$\lambda = 305$
1	λ_{type_1}	6.60	13.20	19.80	6.00	12.00	18.00
	T_{d1}	4.4	8.8	13.3	4.0	8.0	12.1
2	λ_{type_2}	2.18	4.36	6.53	1.80	3.60	5.40
	T_{d2}	1.5	2.9	4.4	1.2	2.4	3.7
3	λ_{type_3}	1.05	2.10	3.15	0.79	1.58	2.36
	T_{d3}	0.7	1.5	2.2	0.6	1.1	1.7
4	λ_{type_4}	0.17	0.34	0.52	1.41	2.82	4.24
	T_{d4}	41.6	83.3	124.9	339.9	679.7	1019.6
System down time T_d		48	97	145	346	691	1037

Table 3: Failure rates (per year) and expected down time (minutes per year) for $\mu_1 = 60, \mu_2 = 3600$.

both models are practically the same. The difference is on average less than 0.5%, which can be explained by rounding off the decimal. Whereas results from Lyu & Mendiratta (1999) provide only the system down time values, Tables 2 and 3 also provide the down time for each recovery procedure.

Two conclusions can be directly drawn from Tables 2 and 3:

- System down time is proportional to the value of the total failure rate; that is also clear from the formulas (4) and (8).
- Increasing the value of the fault detection coverage c_i significantly decreases down time.

The value of the fault detection coverage c_i determines the number of failures that are escalated from the level i to the next level (not to last level directly). Increasing c_i from 0.9 to 0.99 decreases down time several times (6.2 times from 419 min to 68 min for $\mu_1 = 30, \mu_2 = 1800$; 7.2 times from 346 min to 48 min for $\mu_1 = 60, \mu_2 = 3600$). However, the fault detection coverage value depends on the nature of software failures and cannot always be changed by system designers. Furthermore, systems with a bad diagnostic subsystem (which cannot determine failures requiring immediate manual repair) have a greater fault detection coverage value. But if some failures are eventually escalated (step by step) through all levels to the last one, system down time will increase because of the time lost at each level.

There are at least two ways for designers to improve availability of a system:

- Change recovery strategy.
- Improve individual recovery procedures.

Changing the existing recovery strategy by creating new recovery procedures can require significant effort from designers. However, in some cases it is possible to improve availability just by changing the order in which recovery procedures are applied. The segregated failures model can be useful for studying such changing.

An existing recovery procedure can be improved by reducing mean recovery time for the procedure. High recovery time for a specific procedure influences system availability even when the number of failures restored at this level is negligible. Thus, according to Tables 2 and 3 for $c_i = 0.99$, on average only 2.1% of all failures are restored at level 4, i.e. by manual repair. However, the contribution of this level to system down time comes to 86% (85.1% for $\mu_1 = 30, \mu_2 = 1800$ and 86.7% for $\mu_1 = 60, \mu_2 = 3600$). The reason is that, according to Table 1, the mean restoration time for the manual repair is two hundred times more than the mean restoration time for other procedures.

Mean restoration time can be reduced by using better diagnostic equipment, speeding up a delivery of spare parts, etc. Thus, if the mean restoration time for the manual repair is reduced by 20% (from 240 min to 192 min) then, according to (8) and (9), system down time will be reduced on average by 18%.

5 Conclusion

In this paper, we considered fault-tolerant computer systems with several recovery procedures. We continued the earlier investigations described in Vilkomir *et al.* (2005) by investigating the segregated failures model for availability evaluation of such systems. This model provides a simple analytical method of evaluating system availability and can be used as an alternative to Markov chain models. We analysed both approaches and showed that the segregated failures model not only allows us to calculate down time of a whole system but also gives the possibility, using intermediate results of calculations, of evaluating the impact of each recovery procedure on system availability. The model can also be used for the correction of a recovery strategy and improvement of system availability.

We extended the segregated failures model for the situation when the values of implicit rates of failures escalation are known instead of coverage factors. As a case study, a Lucent Technologies Reliable Clustered Computing architecture was considered. Different values of failure restoration rates were considered and their influence on down time was analysed. The values of down time for every type of failures as well as for the whole system were calculated. For this specific case study, the main factor that impacted system availability was the mean restoration time for the manual repair. Thus, reducing this time is an important practical task to improve availability.

The case study shows that the segregated failures model provides a simple, convenient and practical approach for availability evaluation. In future work, we will consider an application of this model to the analysis of different recovery strategies and selection of an optimal strategy for any given system.

6 Acknowledgement

This work was supported by Science Foundation Ireland under SFI Grants 01/P1.2/C009 and 03/CE3/1405.

References

Hoeflin, D.A. & Mendiratta, V.B. (1995), Elementary Model for Predicting Switching System Outage Durations, *in* 'Proceedings of XV International Switching Symposium', Berlin, Germany, 23–28 April, 1995.

Hughes-Fenchel, G. (1997), A flexible clustered approach to high availability, *in* 'Digest of Papers of Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing', FTCS-27, Seattle, Washington, USA, 24–27 June 1997, pp. 314–318.

Ibe O., Howe, R. & Trivedi, K. S. (1989), Approximate Availability Analysis of VAXCluster Systems, *IEEE Transactions on Reliability*, Vol. 38, No. 1, April 1989, pp. 146–152.

Lyu, M.R. & Mendiratta, V.B. (1999), Software fault tolerance in a clustered architecture: techniques and reliability modelling, *in* 'Proceedings of the 1999 IEEE Aerospace Conference', Volume 5, Snowmass, CO, USA, 6–13 March 1999, pp. 141–150.

Mendiratta, V.B. (1998), Reliability analysis of clustered computing systems, *in* 'Proceedings of the Ninth International Symposium on Software Reliability Engineering', Paderborn, Germany, 4–7 November 1998, pp. 268–272.

Sun, H., Han, J.J. & Levendel, H. (2001), A generic availability model for clustered computing systems, *in* 'Proceedings of 2001 Pacific Rim International Symposium on Dependable Computing', Seoul, Korea, 17–19 December 2001, pp. 241–248.

Sun, H., Han, J.J. & Levendel, H. (2003), Availability requirement for a fault-management server in high-availability communication systems, *IEEE Transactions on Reliability*, Volume 52, Issue 2, June 2003, pp. 238–244.

Vilkomir S., Parnas D., Mendiratta V. & Murphy E. (2005), Availability evaluation of hardware/software systems with several recovery procedures, *in* 'Proceedings of the 29th IEEE Annual International Computer Software and Applications Conference' (COMPSAC 2005), IEEE Computer Society, Volume 1, Edinburgh, Scotland, 25–28 July 2005, pp. 473–478.

On pedagogically sound examples in public-key cryptography

Suan Khai Chong **Graham Farr** **Laura Frost** **Simon Hawley**

Clayton School of Information Technology
Monash University
Clayton, Victoria 3800
Australia
Email: {skcho5,gfarr,lauraf}@csse.monash.edu.au, sa_hawley@yahoo.com.au

Abstract

Pencil-and-paper exercises in public-key cryptography are important in learning the subject. It is desirable that a student doing such an exercise does not get the right answer by a wrong method. We therefore seek exercises that are *sound* in the sense that a student who makes one of several common errors will get a wrong answer. Such exercises are difficult to construct by hand. This paper considers how to do so automatically, and describes software developed for this purpose, covering several popular cryptosystems (RSA, Diffie-Hellman, Massey-Omura, ElGamal, Knapsack). We also introduce *diagnostic* exercises, in which all error paths lead to different answers, so that the answer given by the student may suggest the nature of their error. These too can be generated automatically by our software.

Keywords: sound example, diagnostic example, example generator, public-key cryptography, RSA, Diffie-Hellman, Massey-Omura, ElGamal, knapsack.

1 Introduction

Public-key cryptography is now taught in a large number of courses around the world, in response to the rapid development of the subject and its applications since the first papers in the area (Diffie & Hellman 1976, Rivest, Shamir & Adleman 1978). Countless textbooks, articles and lectures take students through the principles and manipulations involved in the best-known public-key cryptosystems (such as the RSA (Rivest, Shamir & Adleman 1978), ElGamal (ElGamal 1985) and Knapsack (Merkle & Hellman 1978) systems) and their relatives (such as the Diffie-Hellman key exchange scheme (Diffie & Hellman 1976) and the Massey-Omura realisation (Massey & Omura 1986) of the Shamir three-pass protocol). To learn how these systems work, it is helpful for students to have exercises on them to do by hand. These exercises will often use small numbers, so that students can see the how the various steps work and fit together without being lost in difficult computation. This is, of course, quite opposite to the demands of practical applications, where large numbers must be used in order to make cryptanalysis difficult.

If a student is to work through a particular exercise themselves, it is important that, as far as possible, they only get the right answer if they use the right method. Students may often have access to the right answer after doing the exercise (e.g., from their lecturer, or the back of a textbook), and if their answer is the same, then their method of calculation is likely to be reinforced in their minds.

In doing public-key cryptography exercises manually, certain errors crop up repeatedly. We enumerate those that are most common in our experience in §§3–4. Of course, it is not possible to give an exhaustive list, and there is always the possibility that a student may make an unpredictable arithmetic slip or some conceptual error we have not anticipated. Nonetheless, the most common and predictable errors are worth taking some account of.

It is not altogether trivial to construct, by hand, exercises with small numbers for which common errors give a wrong answer. For example, suppose we ask a student to do RSA cryptanalysis manually for what is virtually the smallest possible example: $p = 3$ and $q = 5$, so that $n = 15$ and $\varphi(n) = 8$. We find that every $d \in \mathbb{Z}_{\varphi(n)}$ is its own multiplicative inverse. Any exercise constructed with these numbers will have the unfortunate property that a student who mixes up the public and private exponents, and so encrypts instead of decrypting, will still get the right answer. The same problem occurs for $(p, q) = (3, 7)$ or $(5, 7)$, and for other pairs (p, q) of small primes one may still need to choose with care as self-inverse d are reasonably numerous.

With these concerns in mind, we have studied, for the public-key cryptosystems mentioned above, and for certain kinds of exercises based on these systems, the set of exercises that are *sound* in the sense that any set of errors from a given list will lead to a wrong answer. We have constructed a program that generates sound exercises of the required sort, and enables study of the set of possible exercises.

The program may be applied not only to help the teacher construct better exercises, but as an on-line tool for use by students. A student can ask for a sound example to be generated, try to solve it by hand, enter their answer, and be informed whether their answer is right or not.

Applications of this sort suggest another, stronger property of exercises. We say that an exercise is *diagnostic* if all of the methods considered — the right one, and all the wrong ones we have allowed for — give different answers. The idea here is that the answer supplied by the student gives some evidence as to the nature of the error(s) (if any) they may have made (though the evidence is not absolutely conclusive, since a student may make unpredictable arithmetic slips or other errors we have not anticipated). Diagnostic exercises may thus enable some useful online feedback to the student. Our system can also generate diagnostic exercises, where they exist, and assist in study of the set of diagnostic exercises for a given cryptosystem.

The exercises we consider are generally based on carrying out the operations of encryption and decryption in the various public-key cryptosystems. We do not cover all possible tasks that might be set for these cryptosystems, but only ones that illustrate the main manipulations involved in using the systems and for which certain kinds of errors (discussed in §3) keep coming up. The exercises generated by our system can, in any case, often be used as the basis for other exercises of slightly different character.

The concepts and software described here can also be applied to exercises using larger numbers that are designed to be solved with the aid of calculators or mathematical software. Our emphasis is on pencil-and-paper exercises, since students may have more feel for manipulating smaller numbers. Also, exercises with smaller numbers have a lower chance of being sound or diagnostic if chosen at random, so there is a clear role for automated assistance in constructing such exercises.

We assume familiarity with public-key cryptography, particularly the RSA, Diffie-Hellman, Massey-Omura, ElGamal and Knapsack systems (see, e.g., (Welsh 1988)).

This work is based on Summer Studentship projects by Chong (2003–04), Frost (2002–03) and Hawley (2002–03), and especially on Chong's BCompSc Honours project (Chong 2003), all supervised by Farr at the School of Computer Science and Software Engineering, Monash University.

The rest of this paper is organised as follows. In the next section we give more formal definitions of the concepts of sound and diagnostic examples. In §3 we describe what seem to be the main kinds of error in the systems we consider: these occur repeatedly in exercises on all the different systems. In §4 we describe the exercise types themselves and, for each, the exact set of errors we use. In §5 we give an overview of the software, describing the main functionalities present in the system. Some conclusions and suggestions for further work are given in §6.

2 Definitions

In the cryptography exercises we consider, the student is given some input and must calculate the corresponding output. The inputs and outputs are numeric rather than symbolic, and usually belong to finite algebraic systems such as $\mathbb{Z}_n$. Exercises are usually of one of two types: encryption, where the student is given the public key and the message and must calculate the cypher (or any other information that is sent between sender and receiver); and decryption, where the student is given the public key and the cypher (and anything else that is exchanged between sender and receiver) and must work out the original message (or any other secret information shared by sender and receiver).

The aim of such a cryptography exercise is to get the student to practise carrying out some algorithm A_0 for computing the desired output from the given input. The algorithm A_0 may be (slightly) nondeterministic: there might be some flexibility in how the student works out the answer.

We suppose that there is also a set of alternative, incorrect algorithms $A_1, \ldots, A_k$ that each might be used, by some students, to try to solve the exercise.

An exercise X is *sound* if each of the alternative algorithms produces an incorrect answer: $A_i(X) \neq A_0(X)$ for all $i = 1, \ldots, k$.

An exercise X is *diagnostic* if all of the algorithms — correct or incorrect — give different answers: $A_i(X) \neq A_j(X)$ for all $i, j \in \{0, 1, \ldots, k\}$, $i \neq j$.

We assume that the alternative algorithms $A_1, \ldots, A_k$ arise in the following way. Let E be the assumed set of errors that a student might make in executing A_0. Let $f : \{0, 1, \ldots, k\} \rightarrow 2^E$ be a bijection with $f(0) = \emptyset$ and $k = 2^{|E|} - 1$. Let algorithm A_i be the algorithm obtained by trying to execute A_0 but making precisely the errors from the set $f(i) \subseteq E$ while doing so. In this way, each algorithm is identified with some subset of the error set E, with A_0 being the correct algorithm in which no errors are made.

In what follows, the set of alternative algorithms will be specified by giving the error set E.

3 Errors

For all the systems we consider, many of the same basic errors recur, mostly involving choice of modulus, confusion between encryption and decryption, and errors in carrying out the extended Euclidean algorithm (EEA). (Recall that the EEA applied to coprime a and b produces a sequence of triples (δ, x, y) where $\delta = ax + by$, culminating in $(1, u, v)$, where $v \equiv b^{-1} \pmod{a}$. A student might choose u as the inverse instead of v; note that $u \equiv a^{-1} \pmod{b}$. If $v < 0$, the student might ignore the sign and use $-v$. It is also possible that the student will take the extended Euclidean algorithm one step further, to the triple $(0, \pm b, \mp a)$. This is useful as a check, but a student might erroneously take one of $\pm b$ or $\mp a$ as the desired inverse of b, in effect treating the "0-triple" as if it were the important "1-triple".) Most errors seem to arise through one or more of the following:

- confusion over whether to do a calculation mod n or mod $\varphi(n)$ or mod $n-1$ (even students doing RSA, where $n-1$ has no particular role, sometimes work mod $n-1$ as they may recall seeing $p-1$ in other systems, e.g., $\varphi(p) = p-1$ in the Massey-Omura system);
- using a quantity itself instead of its inverse (e.g., e instead of d in RSA);
- taking the wrong element from the EEA calculation;
- ignoring the sign of the element taken from the EEA calculation.

4 Exercises and error lists

In this section, we take five systems — RSA, Diffie-Hellman, Massey-Omura, ElGamal and Knapsack — and, for each, we describe one or two types of exercise and the assumed error list for each exercise type. The exercise types are usually encryption and decryption.

Our decryption exercises usually take the position of a cryptanalyst attempting to recover the message given the cyphertext and public keys. Sometimes we use decryption exercises where the student is given part of the private key as well. There are a couple of reasons why this may be useful at times. Firstly, if that part of the private key is not given, the exercise may be of such a different kind that the errors students make are quite different to the ones we consider here. The exercise may still be worthwhile, but falls outside the scope of the present work. Secondly, pencil-and-paper exercises may use sufficiently small numbers that some particular part of the private key may be easy to guess in practice, so that the real task only begins once that part of the private key is known. (This does not mean that the lecturer necessarily gives that part of the private key when presenting the exercise to students. We treat the relevant part of the private key as known only when designing the exercise.)

For each exercise type, we state, in turn: the information assumed to be *given* to the student; what the student must *find*; the *method* (i.e., A_0) the student should use; and the set of possible *errors* we consider. In our software (§5), the user selects which of these errors belong to the assumed error set E. For the RSA system, we give an example to illustrate the various concepts we have introduced.

4.1 RSA

Encryption
GIVEN: modulus $n = pq$, public exponent $e \in \mathbb{Z}^*_{\varphi(n)}$, and message $m \in \mathbb{Z}_n$.
FIND: cyphertext $c = m^e \bmod n$.
METHOD: find $c = m^e \bmod n$, preferably by fast modular exponentiation.
ERRORS:

(a) use $\varphi(n)$ instead of n as the modulus for exponentiation;
(b) use $n-1$ instead of n as the modulus for exponentiation.

Decryption
GIVEN: modulus $n = pq$, public exponent $e \in \mathbb{Z}^*_{\varphi(n)}$, and cyphertext $c \in \mathbb{Z}_n$.
FIND: $m = c^d \bmod n$, where $d = e^{-1} \bmod \varphi(n)$.
METHOD: factorise n, to find p and q; calculate $\varphi(n) = (p-1)(q-1)$; find $d = e^{-1} \bmod \varphi(n)$ using the EEA; and find $m = c^d \bmod n$, preferably by fast modular exponentiation.
ERRORS:

(a) use $n-1$ instead of $\varphi(n)$ as the modulus when finding e^{-1};
(b) use n instead of $\varphi(n)$ as the modulus when finding e^{-1};
(c) use e instead of d;
(d) negate the inverse e^{-1};
(e) swap the modulus and d when computing d^{-1} (so, with modulus $\varphi(n)$, the student finds $\varphi(n)^{-1} \bmod d$ instead of $d^{-1} \bmod \varphi(n)$);
(f) use $\varphi(n)$ instead of n as the modulus for exponentiation;
(g) use $n-1$ instead of n as the modulus for exponentiation.

Example
Suppose we want an RSA decryption exercise with assumed error set set $E = \{(b),(c),(f)\}$. Let $n = 77$, so that $p = 7$, $q = 11$. If $d = 17$, $e = 53$, $m = 12$, and $c = 45$, then we could give a student the decryption exercise $(n, e, c) = (77, 53, 45)$. The correct method A_0 would find $d = 53^{-1} \bmod 60 = 17$ and then $m = 45^{17} \bmod 77 = 12$. However, this same answer is obtained if error (c) is made, but no others: putting $d' = e = 53$ yields $m' = 45^{d'} \bmod 77 = 12 = m$. This example is therefore unsound. A sound example (for this error set) can be obtained by putting $d = 17$, $e = 53$, $m = 6$, and $c = 62$. This example is not, however, diagnostic, since the incorrect answer $m' = 32$ is obtained either by making error (f) alone or by making errors (c) and (f). The following example may be shown to be diagnostic for E: $n = 161$, $p = 7$, $q = 23$, $d = 13$, $e = 61$, $m = 17$, $c = 80$. Diagnostic examples tend to be much harder to construct than ones that only have to be sound.

4.2 Diffie-Hellman

GIVEN: prime p, primitive root $a \in \mathbb{Z}_p$, $y_1 = a^{x_1} \bmod p$, $y_2 = a^{x_2} \bmod p$.
FIND: $K = a^{x_1 x_2} \bmod p$.
METHOD: *either* find x_1 (i.e., solve the discrete log problem $y_1 = a^{x_1} \bmod p$), then form $K = y_2^{x_1} \bmod p$, *or* find x_2, then form $K = y_1^{x_2} \bmod p$.
ERRORS:

(a) find discrete log mod $p-1$ instead of mod p;
(b) do exponentiation mod $p-1$ instead of mod p.

4.3 Massey-Omura

The Massey-Omura cryptosystem follows the Shamir three-pass protocol.

Encryption
GIVEN: prime p, $x \in \mathbb{Z}^*_{p-1}$, $y \in \mathbb{Z}^*_{p-1}$, m.
FIND: $m^x \bmod p$, $m^{xy} \bmod p$, $m^y \bmod p$.
METHOD: calculate $m^x \bmod p$, $(m^x)^y \bmod p$, and then *either* calculate $m^y \bmod p$ *or* find $x^{-1} \bmod p-1$ and calculate $(m^{xy})^{x^{-1}} \bmod p$. (The latter is more efficient, and is what the sender actually does when using this system. The former is sometimes done by students in exercises, and will still give the right answer if done correctly, although it cannot be done in practice since it involves the sender using information known only to the receiver.)
ERRORS:

(a) do exponentiation mod $p-1$ instead of mod p, at any stage;
(b) find $x^{-1} \bmod p$ instead of mod $p-1$;
(c) negate the inverse x^{-1};
(d) swap the modulus and x when computing x^{-1} (so, with modulus $p-1$, the student finds $(p-1)^{-1} \bmod x$ instead of $x^{-1} \bmod p-1$);
(e) use x instead of x^{-1}.

Decryption
Here we consider the somewhat artificial situation in which the cryptanalyst is given (or has found) each party's private information, except the message, and must only recover the message.
GIVEN: p, $x \in \mathbb{Z}^*_{p-1}$, $y \in \mathbb{Z}^*_{p-1}$, $m^{xy} \bmod p$.
FIND: m.
METHOD: *either* find $x^{-1} \bmod p-1$, calculate $m = (m^{yx})^{x^{-1}} \bmod p$, find $y^{-1} \bmod p-1$ and calculate $m = (m^y)^{y^{-1}} \bmod p$.
ERRORS:

(a) find inverse mod p instead of mod $p-1$;
(b) use x instead of x^{-1} (or y instead of y^{-1});
(c) negate the inverse;
(d) swap the modulus and x when computing x^{-1} (or similarly when computing y^{-1});
(e) do exponentiation mod $p-1$ instead of mod p.

4.4 ElGamal

Encryption
GIVEN: prime p, primitive root $a \in \mathbb{Z}_p$, $y = a^x \bmod p$, $k \in \mathbb{Z}_{p-1}$, $m \in \mathbb{Z}_p$.
FIND: $c = (a^k \bmod p, Km \bmod p)$ where $K = y^k = a^{xk} \bmod p$.
METHOD: find $a^k \bmod p$, $K = y^k \bmod p$, form $Km \bmod p$; $c = (a^k, Km) \bmod p$.
ERRORS:

(a) do exponentiation $a^k \bmod p-1$ instead of p;

(b) do exponentiation $y^k \bmod p-1$ instead of p;

(c) do the final multiplication mod $p-1$ instead of mod p.

Decryption
GIVEN: p, a, $y = a^x \bmod p$, $a^k \bmod p$, $Km \bmod p$ where $K = y^k \bmod p = a^{xk} \bmod p$.
FIND: m.
METHOD: find K as in Diffie-Hellman, find K^{-1} mod p, find $m = K^{-1}(Km) \bmod p$.
ERRORS:

(a) in Diffie-Hellman subproblem, to find K: find discrete log mod $p-1$ instead of mod p;

(b) in Diffie-Hellman: do exponentiation mod $p-1$ instead of mod p;

(c) find K^{-1} mod $p-1$ instead of mod p;

(d) use K instead of K^{-1} (i.e., as its own inverse);

(e) negate the inverse;

(f) swap the modulus and K when computing K^{-1};

(g) do the final multiplication mod $p-1$ instead of mod p.

4.5 Knapsack

The Knapsack cryptosystem is considered insecure, but is still useful for teaching purposes. We only consider decryption exercises. Encryption, while also good for students to do, does not have much potential for the kind of errors we consider.

Decryption
This exercise may seem somewhat artificial: the cryptanalyst is assumed to know the private multiplier w. Without w, on the face of it the student just has to solve an ordinary (nonsuperincreasing) subset sum problem manually, which involves different kinds of potential errors to the ones considered in this paper (though it is a good exercise for them to do). Also, for small manual exercises, w is often easily deduced by guessing the smallest term or two of the private superincreasing sequence and observing how they are related to the corresponding terms of the public sequence.
GIVEN: N, $w \in \mathbb{Z}_N^*$, public sequence $(a_i)_{i=1}^n$, cypher block $c = \sum_{i=1}^n a_i m_i$, where the m_i are the message bits.
FIND: $m = (m_i)_{i=1}^n$.
METHOD: find w^{-1} mod N, find the private superincreasing sequence $(x_i)_{i=1}^n$ by $x_i = w^{-1}a_i \bmod N$, find $w^{-1}c \bmod N$, and find the m_i by solving the superincreasing subset sum problem $w^{-1}c = \sum_{i=1}^n x_i m_i$.
ERRORS:

(a) find w^{-1} mod $N-1$ instead of mod N;

(b) use w instead of w^{-1};

(c) negate the inverse;

(d) swap the modulus and w when computing w^{-1};

(e) do the final multiplications (to obtain the x_i) mod $N-1$ instead of mod N.

5 Software

We have written software for generating examples with the properties discussed in §4 (Chong 2003). This section describes the main features of the system and related functionalities that give users control of the system. We begin by describing the main modes of operation which present different ways in which the system can be used. We then briefly discuss the selection of error paths in the system (discussed in §3) and of how the information is displayed.

Source code for the software mentioned in this paper includes C programs for the example generators and Java programs for the web interface. The C files for the example generators can be obtained from:
`http://www.csse.monash.edu.au/~skcho5/CryptoTools/generators.zip`
and the Java interface files can be downloaded from:
`http://www.csse.monash.edu.au/~skcho5/CryptoTools/web.zip`

The software can be run online by using any browser that supports Java 1.1 or above[1] on the webpage
`http://www.csse.monash.edu.au/~skcho5/CryptoTools/Gui.html` .

5.1 Modes of operation

Three main modes of operation are provided in the system - Interactive mode, Random mode and Calculate mode. Each of those modes is now described in turn.

5.1.1 Interactive mode

Interactive mode provides the ability to check whether an example supplied by the user is unsound, sound or diagnostic. First, the user is required to provide inputs that specify an example, typically the public key values, private key values and the message. The system then validates these inputs. For instance, for an RSA example, the value of the exponent e must be in $\mathbb{Z}_{\varphi(n)}^*$, and the user is warned if a self-inverse exponent is chosen.

After validation, the system can check which category the examples falls into. This mode is useful as a quick check for a manually generated example, for instance, an exercise in a textbook. However, for automatic generation of examples, random mode should be used instead.

5.1.2 Random mode

Random mode is the mode by which the system performs its main function: automatic generation of random sound or diagnostic examples. At each iteration, random inputs are chosen. These must be small, and must satisfy certain validity tests (as mentioned in §5.1.1). The inputs together give a random example, from which the set of paths is constructed. On comparison of the paths, the system determines if the example is sound or diagnostic. If the example generated is not of the desired type, another iteration is attempted, and so on, until an example of the desired type is found.

Other supplementary modes such as no-filtering and target mode can be used with random mode. No-filtering mode removes the restriction on input sizes, so the example generated might use numbers that are too large for pen and paper exercises (though might still be suitable for exercises using calculators or mathematical software). It also increases the chance of a generated example being sound or diagnostic. Target mode makes the system randomly generate examples with user specified target values for some of the inputs. (The actual selected input will be within 5% of the specified target.)

[1]Support for Macintosh browsers is currently limited to Netscape versions only.

5.1.3 Probability calculation mode

Probability mode enables the user to study how common sound or diagnostic examples are, among all possible examples. For each input, the program either takes a value from the user or loops over all possible values. The proportions of sound and diagnostic examples are tabulated and the probabilities of these example types are shown. This capability allows us to study what kinds of inputs give a higher proportion of sound or diagnostic examples and also, the minimum input sizes for constructing sound or diagnostic examples.

5.2 Path selection

Path selection allows users to choose, from a list of possible errors (being just those error lists given in §4), which errors are to be included in the assumed error set E. This reflects the observation that some errors are more likely than others and that the error set may need to be adapted to the different needs of different groups of students.

5.3 Path display

In our implementation, a *path* is a sequence of numbers obtained by carrying out the successive steps of an algorithm, where path i is the path obtained from algorithm A_i (refer §2). A collection of paths gives us a *path table* where the top row of the table will always be the correct path and the remaining rows account for all other paths corresponding to all the subsets of E. An alternative path display is in the form of a *path tree* where the paths displayed are grouped by similar error subsets, which enables the user to trace the consequences of errors. The program allows the user to choose either a path table or a path tree display.

5.4 Using the software

The programs can be run with a variety of command-line options. Full details may be found in the man pages or in our technical report (Chong, Farr, Frost & Hawley 2004). We briefly describe the operation of the program `rsa`, which generates examples for the RSA system.

This program may be run in interactive mode using the command

```
rsa -e
```

It outputs a list of the available primes and prompts the user for p and q. Subsequent interaction allows the user to choose the other numbers used by RSA, leading to a particular choice of public and private keys, which the user may accept or reject (and try for another). The user is advised which exponents in $\mathbb{Z}^*_{\varphi(n)}$ are self-inverse, but is not prevented from choosing such a value. Once the problem is fully specified, the program outputs all possible error paths for the decryption problem, advising the user of whether or not the example chosen is sound. Interactive mode allows study of the error paths of any example, whether sound or unsound, but does not generate examples for the user.

Random mode is the main mode of the program. Suppose the user wants a randomly generated sound example with $n \simeq 77$, using the same error set as in our example in §4.1: $E = \{(\mathrm{b}), (\mathrm{c}), (\mathrm{f})\}$. Then the user enters

```
rsa -g -t 77 -s0110010
```

The successive bits in the `-s` option are used to switch on, or off, the corresponding errors from (a)–(g) in §4.1. The program might then randomly choose the sound example given in §4.1: $p = 7$, $q = 11$, $d = 17$, $e = 53$, $m = 6$, $c = 62$. In this case the student would be given the public key $(n, e) = (77, 53)$ and the cyphertext $c = 62$, and asked to find the message m. The program's output includes a table giving the results of all error paths under our assumed error set E:

```
Path |  d       phi_n   mod     m'
----------------------------------
  1  |  17      60      77      6
  2  |  17      60      60      32
  3  |  16      77      77      15
  4  |  16      77      60      16
  5  |  53      -       77      13
  6  |  53      -       60      32
```

The first column here gives the path number. The second gives the value of d used. This corresponds to which, if any, of errors (b) and (c) are made, with $d = 17$, 16 or 53 according as neither, (b) only, or (c) only is made. Note that errors (b) and (c) will not both be made, and that this column is not affected by whether or not error (f) is made. The third column gives the modulus used for finding $d = e^{-1}$, which should be $\varphi(n) = 60$, but will be $n = 77$ if error (b) is made, and is inapplicable if error (c) is made. The fourth column gives the modulus used for finding $m = c^d$, which should be $n = 77$, but will be $\varphi(n) = 60$ if error (f) is made. The fifth column gives the message found.

Path 1 in the above table is the correct path, and the remaining paths correspond to different subsets of the error set:

Path	Errors made
1	none
2	(f)
3	(b)
4	(b), (f)
5	(c)
6	(c), (f)

The output concludes with a brief remark that this example is not diagnostic and a summary of the amount of searching done before this example was found.

Generation of random diagnostic examples works similarly, with option `-d` instead of `-g`. However, it may be necessary to use larger values of n, or smaller sets of possible errors, otherwise it may be too slow, due to the rarity of such examples.

Probability calculation mode can be employed using

```
rsa -c
```

The user is prompted successively for values of n, e and m. For each of these, the user may enter either a number or the character '`r`'. If the latter, the program will examine all possible values from some appropriate range. Once all three entries have been made, the program examines all possible examples with the given values, or ranges of values, for n, e and m, and determines the numbers of these examples that are sound or diagnostic. By default, all errors in E are permitted. If the user wants a more restricted set, then these can be specified as above. For example, to use the error set $E = \{(\mathrm{b}), (\mathrm{c}), (\mathrm{f})\}$, the user enters

```
rsa -c -s0110010
```

Suppose the user enters $n = 77$ and then enters '`r`' for both e and c. The program reports that it examined 159 different values of m and 40 different values of e (equivalently, of d), and that of the examples thus determined, 1950 were unsound, 4410 were sound, and 2220 were diagnostic, giving probabilities of about 0.31, 0.69 and 0.35 respectively.

The programs may also be run using the web interface mentioned in §5.

6 Conclusions

We have investigated pedagogically sound examples in public-key cryptography and constructed software to generate and study such examples. Our tools aid the task of constructing good exercises for students by automatically generating examples that are sound or diagnostic, where with the absence of such tools, arbitrarily generated examples have a high chance of being unsound.

Future extensions to the tools could include creating an error feedback learning tool for students. The learning tool could use the ideas and software detailed in this report to give effective feedback on errors that students might make. The tool could also be extended to collect answers submitted by students and so carry out a more systematic study of errors that students make.

Some errors are more likely than others. If some estimates of the probabilities of the various errors were known (perhaps from the work envisaged at the end of the previous paragraph), it would be possible to generate exercises for which the probability of getting the right answer by a wrong path is bounded above by some small positive constant.

The ideas of this paper and their implementation could also be applied to other cryptosystems and possibly other teaching problems.

References

Chong, S. K. (2003), Cryptographic teaching tools, BCompSc Honours, Monash University, Clayton, Australia.

Chong, S. K., Farr, G. E., Frost, L., & Hawley, S. (2004), Pedagogically sound examples in public-key cryptography, Technical Report No. 2004/155, School of Computer Science and Software Engineering, Monash University.

Diffie, W. & Hellman, M. E. (1976), 'New directions in cryptography', *IEEE Trans. Inform. Theory* **IT-22** (6) 644–654.

ElGamal, T. (1985), 'A public key cryptosystem and a signature scheme based on discrete logarithms', *IEEE Trans. Inform. Theory* **IT-31** (4) 469–472.

Massey, J. L. & Omura, J. K. (1986), Method and apparatus for maintaining the privacy of digital messages conveyed by public transmission, U.S. Patent number 4,567,600.

Merkle, R. C. & Hellman M. E. (1978), 'Hiding information and signatures in trapdoor knapsacks', *IEEE Trans. Inform. Theory* **IT-24** (5) 525–530.

Rivest, R. L., Shamir, A., & Adleman, L. M. (1978), 'A method for obtaining digital signatures and public-key cryptosystems', *Communications of the ACM* **21** (2) 120–126.

Welsh, D. (1988), *Codes and Cryptography*, Oxford.

Towards Security Labelling

Chuchang Liu[†] **Mehmet A. Orgun**[‡]

[†]Information Networks Division
Defence Science and Technology Organisation
PO Box 1500, Edinburgh, SA 5111, Australia
Chuchang.Liu@dsto.defence.gov.au

[‡]Department of Computing, Macquarie University
Sydney, NSW 2109, Australia
mehmet@ics.mq.edu.au

Abstract

Security labels are applied for numerous reasons, including the handling of data communicated between open systems. The information contained within a security label can be utilised to perform access control decisions, specify protective measure, and aid in the determination of additional handling restrictions required by a communications security policy. This paper concerns the issues regarding security labelling in open systems. We propose a security labelling framework for such systems; and further, based on this framework, we develop a mechanically checkable model for security labelling systems and discuss its implementation issues. This model provides a functional base for future design and implementation of security labelling systems.

Keywords: open systems, security policy, security labelling, label validation.

1 Introduction

The systems considered here are called open systems. An open system is viewed as a set of one or more computers, associated with software, peripherals, terminals, human operators, physical processes, information transfer means, *etc.*, that forms an autonomous whole capable of processing and/or transferring information. A real open system complies with certain requirements in its communication with other systems. Open System Interconnection Environment (OSIE for short) can be regarded as an abstract representation of the set of concepts, elements, functions, services, protocols, *etc.*, as defined by the Basic Reference Model proposed by ISO (the International Organization for Standardization) (ISO/IEC 7498-1 1994), and those specific standards derived from ISO, which enable communications among open systems.

Security labelling as an element of the OSIE is one of mechanisms that provide data security. Data security is the set of measures taken to protect data from unauthorized or accidental modification, destruction, or disclosure. Security labelling plays an important role in enforcing security policies. However, we note that security labelling itself does not provide sufficient data security, it needs to be complemented by other security mechanisms.

Security labels (Internet CIPSO Working Group 1993) applied in open systems convey information used by protocol entities to determine how to handle data transferred between systems. Information contained within a security label can be utilised to control access, specify protective measures, and determine additional handling restrictions required by a communication security policy. Security labelling should be supported by their protocols in communications among open systems. In data communication protocols, security labels provide a support to the protocol processing for correctly handling the data transferred between two systems (Housley 1993). Here handling means the activities performed on data such as collecting, processing, transferring, storing, retrieving, disseminating, and controlling.

Data security includes data integrity and data confidentiality. The data integrity is about protection from modification, destruction, and disclosure. In computer systems, this is protection from writing and deleting. With data integrity, Biba (Biba 1977) proposed a model that includes security labels. The Biba model specifies rule-based controls for writing and deleting necessary to preserve data integrity, and it also specifies rule-based controls for reading to prevent a high integrity process from relying on data that has less integrity than the process. The data confidentiality is about protection from disclosure. In computer systems, this is protection from reading. With data confidentiality, Bell and LaPadula (Bell & LaPadula 1976) defined a model that includes security labels. The Bell-LaPadula model specifies rule-based controls for reading necessary to preserve data confidentiality, and it specifies rule-based controls for writing to ensure that data is not copied to a container where confidentiality can not be guaranteed. In both the Biba and the Bell-LaPadula models, the security label is an attribute of the data. In general, the security label associated with the data remains constant. Housley (Housley 1993) considered the problem of relabelling which often appears as the result of some entity handling the data among open systems.

The security label as an attribute of data should be bound to the data. When data moves among open systems, the integrity security service is generally used to accomplish this binding. If the communications environment does not include a protocol providing the integrity security service to bind the security label to the data, then the communications environment should include other mechanisms to preserve this binding.

The notion of security label also appears in security-typed languages, which have recently been proposed to enforce security properties including confidentiality and integrity by type checking (Heintze

& Riecke 1998, Myers & Liskov 2000, Zdancewic *et al.* 2001). In security-typed languages, types are extended with security labels to enforce information flow control, but the sort of labels are usually applied only to denote security classes associated with users and the resources that programs access (Zheng & Myers 2004).

The security labels we are concerning in this paper are generic. Such labels have a standard form, such as the one of the standard security label defined in (FIPS188 1994), which is specifically applied for information transferring. We endeavour to provide a formal methodology for modelling security labels and labelling systems. The methodology would support the design and implementation of such systems. We do not intend to discuss a practical labelling system, but the methods and techniques for modelling security labels and several important issues regarding the implementation of a labelling system will be presented. Our formal description for security labelling does not discuss the physical labelling of information or storage media and information displayed on a computer screen or other peripherals. Labelling of information stored in internal memory and storage media (e.g. hard disks, compact disks, magnetic tapes, etc.) is also outside of the scope of this paper although similar techniques can be applied. The protection of data in transit and their associated labels along with the binding between the data and the labels is the responsibility of the communications protocols involved in the transfer and therefore not discussed in this paper. Although in the discussion section we give a consideration to threats and some technical discussions on preventing attacks, the compliance with our approach does not provide assurance of the suitability of an implementation for the protection of data according to specific security policies. That assessment must be made through the appropriate evaluation and certification processes.

In this paper, we first discuss a framework of security labelling, in which security domains, security objects, security classifications, caveats, and the security policy as the basic elements involved in security labelling are formally defined. Then, based on the framework, we propose a model for security labelling, which is mechanically checkable and can be easily handled. We discuss implementation issues for this model, including a representation of abstract syntax for a security labelling system, which gives suitable syntactic constructs for conveying security label information, and the basic elements (mechanisms) of the system. We also propose a method for the label validation based on the model. The model proposed in this paper provides a functional base for future design and implementation of security labelling systems. It can be easily modified according to specific security requirements. Our approach is therefore very general, it would be useful for guiding the developer towards a design of a labelling system and its implementation, and it may also help the user to understand and analyse whether such a system satisfies security properties required.

The paper is structured as follows. Section 2 discusses several preliminary notions related to security issues of open systems. Section 3 proposes a security labelling framework for such systems. In Section 4, we present a model for security labelling systems. Section 5 discusses implementation issues with this model. Section 6 contains some further considerations for our model, and Section 7 concludes the paper.

2 Preliminary Notions

Considering those elements that are essential to the security labelling for open systems, we first give the following (informal) definitions:

- **Security policy**: A set of criteria for the provision of security services, which defines and constrains the activities of a data processing facility in order to maintain security conditions for systems and data.
- **Security level**: A hierarchical indicator of the degree of sensitivity to a certain threat. Any particular security level implies a specific level of protection according to the security policy being enforced.
- **Security domain**: A collection of entities, to which applies a single security policy executed by a single authority where an entity can be regarded as a *subject* or an active *agent* (a person, a computer or something else) in an open system.
- **Security information object**: A resource, tool, or mechanism used to maintain a condition of security in a computerized environment. The class of objects are defined in terms of attributes they possess, operations they perform or are performed on them, and their relationship with other objects.

Security labels contain security tags or tag sets to carry security-related information applied for the protection of information exchanged among open systems. According to (ITU-T Recommendation X.841 2000), the aspects of security expressed by a security policy, indicated in a security label, include the level of protection to be given to data stored in a system, who is authorized to access data, processes or resources, security markings shown on any display or print of the material, routing and enciphering requirements for data transmitted between systems, requirements for protection against unauthorized copying, and so on. When data held on an system or when it transmitted electronically between systems, the data are labelled to indicate the security compartment to which the data belongs and thus how the data to be handled for security. Thus, we further need to define

- A *security tag* is an information unit containing a representation of certain security-related information (e.g., a restrictive attribute bit map). A named *tag set* is the field containing a *TagSetName* and its associated set of security tags. The *TagSetName* can be numeric identifier associated with a set of security tags.
- A *security label*, consisting of one or more security tag sets, is a marking bound to a security information object; it names or designates the security attributes of that object.
- A *security labelling system* is a system (or software) used to generate security labels for information objects that need to be labelled.

Formal definition for these notions will be given later.

3 Security Labelling Framework

Assume that there are protected documents which are regarded as security information objects and need to be appropriately labelled such that each document is correctly disseminated utilising security mechanisms. Usually, a labelling system is employed to perform such tasks. The labelling system in general depends on the security labelling framework, which is based on security domains (subjects), security information objects (or simply objects), security classification, caveats, and security policies. Formally, we define the following notations:

$\mathcal{S} = \{s_1, \ldots, s_m\}$ – a finite *subject* set;
$\mathcal{O} = \{o_1, \ldots, o_n\}$ – a finite *object* set;
$\mathcal{L} = \{l_1, \ldots, l_r\}$ – the set of *security levels* with the partial order "$\le$" defined;
$\mathcal{C} = \{c_1, \ldots, c_s\}$ – the set of *caveats* (related to security tags); and
$\mathcal{P} = \{p_1, \ldots, p_k\}$ – the *policy set*.

In reference to security levels, caveats and the policy set, more details are given for further discussions.

The set of security levels, $\mathcal{L}$, is associated with a partial ordering relation, $\le$. We call $(\mathcal{L}, \le)$ a *security classification system*. $(\mathcal{L}, \le)$ is called a linear hierarchy classification system if, for all $l_i, l_j \in \mathcal{L}$, $l_i \le l_j$ when $i \le j$. A specific linear hierarchy classification system is shown in Figure 1(a), which is commonly used and consists of five security levels with the relation that *unclassified* < *restricted* < *confidential* < *secret* < *top_secret*.

The classification system $(\mathcal{L}, \le)$ may be formed as a (finite) lattice. In such a case, there exists a unique element, say l_0, such that for all $i, l_0 \le l_i$, and, symmetrically, there exists a unique element, say l_r, such that for all $i, l_i \le l_r$. Thus, l_0 and l_r are called the minimum element and the maximum element of this lattice respectively or, accordingly, the lowest security level and the highest security level of the classification system. It is not difficult to show that the two security classification systems in Figure 1 are lattices. For both of the systems, the lowest security level is *unclassified* and the highest security level is *top_secret*.

We argue that it may be possible to implement a single security labelling system which encompasses multiple classification systems based on the lattice classification method. In fact, when the security classification system is a lattice, any path from the lowest security level to the highest level forms a single linear hierarchy classification system, see Figure 1(b).

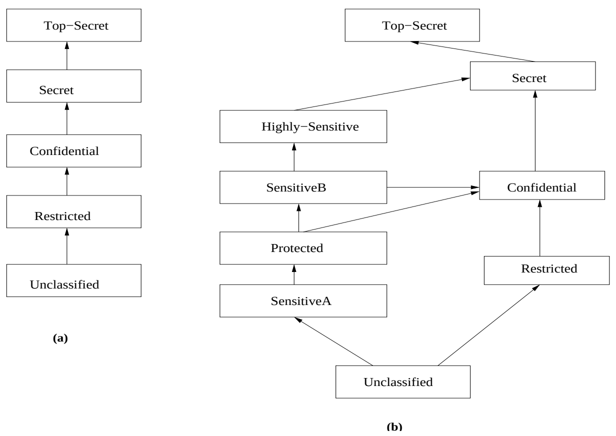


Figure 1: Security Classification Systems

A caveat appearing in a label contains information related to security tag type, tag names and scope to which the object marked with the label would be delivered. A label for a particular object may have several caveats. Due to the assumption we have made that each object has only one label, if a caveat appears in a label of an object, we may simply say that the object has the caveat. In contrast, if a subject is contained within the scope of a caveat, we may say that the subject belongs to the caveat. Further discussion about caveats will be given later.

Policies $p_i, i = 1, \ldots, k$, are defined as boolean functions, such as:

$$read\_deny: \mathcal{S} \times \mathcal{O} \rightarrow \{true, false\}$$

It is a predicate defined over $\mathcal{S} \times \mathcal{O}$. $read\_deny(s, o)$ means that the subject s is not allowed to read the object o (or, the request of s for reading o is denied).

Note that, although the security levels have created the possibility of allowing some object to be kept secret from some subjects, the particular policy may cause an abnormal case where the subject must be kept out from some object(s) that would be delivered to it in the usual case. For instance, assume that $john$ and $mary$ have the same security level *restricted*, usually both of them may be allowed to access restricted documents. However, it is possible that there is a specific policy by which a restricted document d is not allowed to be delivered to $john$ but $mary$ can receive it.

We now propose the framework for security labelling as follows.

Definition 1 *Given an open system G, let $\mathcal{S}$ be the subject set, $\mathcal{O}$ the object set, $(\mathcal{L}, \le)$ the security classification system within G, $\mathcal{C}$ the set of caveats (see below for the definition of caveats), and $\mathcal{P}$ the policy set. Then we call $\langle \mathcal{S}, \mathcal{O}, (\mathcal{L}, \le), \mathcal{C}, \mathcal{P}, F\rangle$ the security labelling framework for the system G, where F is a pair (f_l, f_c) such that f_l is a total function from $\mathcal{S} \cup \mathcal{O}$ to $\mathcal{L}$ and f_c is a total function from $\mathcal{S} \cup \mathcal{O}$ to $2^{\mathcal{C}}$:*

$$f_l: \quad \mathcal{S} \cup \mathcal{O} \rightarrow \mathcal{L}.$$
$$f_c: \quad \mathcal{S} \cup \mathcal{O} \rightarrow 2^{\mathcal{C}}.$$

Intuitively, in this definition $f_l(x)$ is the classfication level assigned to the object or subject x, and $f_c(x)$ is the set of caveats which x (as an object) possesses or x (as a subject) belongs to.

4 A Model for Security Labelling Systems

Security labelling systems are used to generate security labels for the objects that need to be labelled in open systems. In this section, we present a model for security labelling, then discuss implementation issues with this model in the next section.

4.1 Label Format

For simplification of our discussion, we assume that each object will be only assigned one label and every label has only one tag set. In practice, a label may contain information regarding security label identifier and security label length etc. Without loss of generality, we define that a label has the following format:

$$[\mathit{PolicyID}, \mathit{Classification}, \mathit{Caveats}]$$

This format complies with X.841 structures (ITU-T Recommendation X.841 2000). Here the field *PolicyID* is the object identifier for the policy which is applied, the field *Classification* is the security level of the object to be labelled, and the field *Caveats* is a sequence of caveats. A caveat has a format as follows:

$(TagType, TagName, \_)$

where "_" is a field associated with TagName and may be used to contain caveat qualifiers. If no qualifiers are required, then this is deemed by the specification to have the value "NULL".

In X.841, there are five basic security tag types introduced and used for labelling systems: *restrictive bit map* (REST), *Enumerated* (ENUM), *Range* (RANG), *permissive bit map* (PERM), and *Informative* (INFO). The first four types contain the type, one or more non-negative integers and, for the types REST and PERM, also a bit string. The non-negative integer conveys a security level. The type INFO, a free form type, is intended as a wild-card tag type that may carry any user-defined type of data appropriate for use with the protocol handling the labels. These tag types are used to maintain compatibility with a labelling scheme for non-OSI communication systems (Internet CIPSO Working Group 1993). Therefore, in our label format, *TagType* is one of the five types.

The field *TagName* gives the name of the tag such as, for example, *SendTo*, and *DeptOnly*. The meanings of these names are given in advance at the stage of the design for a labelling system, for instance, we assume that *SendTo* stands for "this document is sent to" and *DeptOnly* for "Only department members can access this document". Two caveats as examples are given as follows:

$$Caveat_1 = (PERM, SendTo, \{A_1, \ldots, A_n\})$$
$$Caveat_2 = (REST, DeptOnly, \_)$$

Further investigating caveats, we note that, for example, if caveat $Caveat_1$ is contained in a label bound to an object, then it in fact implies that, based on this caveat together with the consideration of the security level (say L) given in the label, this object can be delivered to only those agents in $\{A_1, \ldots, A_n\}$ whose security level is equal to, or higher than, L. We call the set consisting of all those agents the *scope* of the caveat $Caveat_1$ with L, or simply, the scope of $Caveat_1$, denoted $Scope_L(Caveat_1)$. Similarly, we may have $Scope_L(Caveat_2)$ that consists of all those department members whose security level is equal to, or higher than, L.

In this view, for any given single caveat, we are able to identify its scope based on the form (definition) of the caveat together with the consideration of the security level given in the label. Thus, we can recursively define the scope of a sequence of single caveats, as follows:

$Scope_L(Caveats) = Scope_L(C)$, *if* $Caveats = C$; and
$Scope_L(Caveats) = Scope_L(C_1) \cap Scope_L(C_2 \ldots C_n)$,
if $Caveats = C_1 C_2 \ldots C_n$.

where $C, C_i (i = 1, \ldots, n)$ are single caveats.

4.2 The Mechanically Checkable Model

Based on the security labelling framework, we give the formal definition of a security labelling system model as follows:

Definition 2 *Let $\Theta = \langle \mathcal{S}, \mathcal{O}, (\mathcal{L}, \leq), \mathcal{C}, \mathcal{P}, F \rangle$ be a security labelling framework for a given open system G. A security labelling system model based on the framework is a pair (Θ, ν), where ν is called the labelling function that assigns a label to each object. That is, in this model, for any object $o \in \mathcal{O}$, we have the security label $\nu(o) = [P, L, C_1 \ldots C_k]$ bound to the object, where $P \in \mathcal{P}$, $L = f_l(o)$, and $\{C_1, \ldots, C_k\} \subseteq f_c(o)$, We simply call (Θ, ν) a security labelling system for the system G.*

Intuitively, a security labelling system for a given (open) system consists of the security labelling framework for that system and a labelling function consistent with this framework. In other words, the labelling system is determined by the security labelling framework and a labelling function consistent with this framework.

In our model, the components of the label bound to an object are definitely determined. Firstly, L is explicitly defined by the function f_l of the framework Θ. We now consider how to obtain other two components of the label for a given object. Labels alone are not sufficient to ensure the security of information. The security policy as well as the caveats that applies to the information needs to be enforced in open systems while the labelled information is within the scope of their control. All the organizations, individuals and IT systems that process an item of information are presumed to know the security policy and the caveats for that information. Anyone who want to exchange information with others needs to establish trust in one another to be satisfied that information will be handled according to agreed security policies. This trust is usually established through a formal agreement. Therefore, for any information object, which security policy is applied and what caveats need to be marked in the label of this object should be directly derived from such agreements.

This is a mechanically checkable model. In other words, given the correctness of the security labelling framework for a system, the consistency of the labelling function to the framework is mechanically checkable based on this model. The consistency checking can in fact be easily performed through the label validation process. We will give a detailed discussion on label validation in the next section.

4.3 Remarks about the Model

If a labelling system does not provide sound labels that satisfy a certain security property, then it cannot be regarded as a correct system. In particular, if the security policy within a label is not satisfied, then the label is useless. For verifying whether or not the security policy is satisfied within labels, there is a need to provide a measurement method by which the satisfaction of a policy is easy-checkable.

We now give a formal definition of the *security domain* as follows:

Definition 3 *Let p be a policy and o an object, then the security domain related to p and o, denoted $SD(p, o)$, is defined by $SD(p, o) = \{s | p(s, o)\}$.*

With the access control policy, for example, the security domain explicitly or implicitly gives the domain that contains all individuals (subjects) who are allowed to access the object o under the policy p. While the security label bound to an object provides the actual domain that contains those individuals who may finally be able to touch (or have) the object according to this label. Such an actual domain depends on the label bound to the object. Formally, we have

Definition 4 *Let α be the label bound to an object o. The touch domain of the object o based on α, denoted $TD(\alpha, o)$, is defined by $TD(\alpha, o) = \{s | can\_have(s, o)$ based on $\alpha\}$, where $can\_have(s, o)$ means that the subject s may finally be able to touch (or have) the object o.*

Furthermore, we can give

Definition 5 *Let $\alpha = [P, L, C_1 \ldots C_k]$ be a label bound to an object o within the model defined above, we say that the label α violates the policy P if there is a conflict between the security domain $SD(P, o)$ and the touch domain $TD(\alpha, o)$. In other words, if there exists $s \in TD(\alpha, o)$ but, in other hand, formula $\neg can\_have(s, o)$ can be derived from the fact that $SD(P, o)$ is the security domain related to P and o, then the label α violates the policy P.*

As an example, let us consider the policy be_secret and a file $d1$, assume $SD(be\_secret, d1) = \{s | be\_secret(s, d1)\}$ consists of all individuals who must be kept out from the file $d1$. Assume $\nu(d1) = [P, L, C_1 \ldots C_k]$ where P is the policy be_secret and $TD(\nu(d1), d1)$ contains an element belonging to $SD(be\_secret, d1)$, then there is a conflict and, therefore, $\nu(d1)$ violates the policy P.

Definition 6 *Given a model (Θ, ν), we say that a label $\nu(o) = [P, L, C_1 \ldots C_k]$ is security-valid, if $\nu(o)$ does not violate the policy P.*

Assume agents who control a label as well as the object to which the label is bound always honestly and correctly apply the label to handle the object, then the following formula should be true: for a label $\nu(o) = [P, L, X_1 \ldots X_k]$, we have

$$TD(\nu(o), o) = Scope_L(X_1 \ldots X_k) \qquad \text{(A)}$$

Definition 7 *A model (Θ, ν) is checkable iff there is an algorithm which can be applied to determine whether or not any label is security-valid.*

On this view, our model is mechanically checkable. In fact, based on definitions 4 and 5 and formula (A), in order to check whether a label $\nu(o) = [P, L, C_1 \ldots C_k]$ is security-valid, what we need to do is, for all $s \in Scope_L(X_1 \ldots X_k)$, to check whether s is not allowed to touch (have) the object o, according to the security domain $SD(P, o)$. Considering that $SD(P, o)$ and $Scope_L(X_1 \ldots X_k)$ are all computable, there is no difficulty to construct such an algorithm applied to determine whether or not a label is security-valid (see Section 5.4).

5 Implementation Issues

The implementation of a labelling system within our model involves many technical aspects, including the representation of syntax, the techniques for binding security labels to objects, and the techniques applied for reasoning, especially for label validation, *etc.* It also rely on the availability of a registration service to assign $TagSetName$ and serve as the repository of the semantics, special handing rules, and other details required for the use of security policy-specific label sets. When security labels are specific to a particular application, it may also be related to a specific application protocol. Implementing a practical security labelling system is not in the scope of this paper. Instead, we only discuss several technical problems, and point to some essential elements for implementing such systems.

In this section, we first introduce the syntax and those basic mechanisms needed for implementing a security labelling system, then discuss two specific aspects – binding security labels to objects, and label validation.

5.1 Abstract Syntax

Based on our model given above, a representation of abstract syntax for the labelling system can be obtained as follows:

$$\begin{array}{rcl}
Label & ::= & [PID, Cls, Cavts] \\
PID & ::= & p_1 \mid \ldots \mid p_t \\
Cls & ::= & l_1 \mid \ldots \mid l_r \\
Cavts & ::= & Caveat \mid Caveat\ Cavts \\
Caveat & ::= & (TagType, TagName, Domain) \\
TagType & ::= & \text{REST} \mid \text{PERM} \mid \text{ENUM} \mid \ldots \\
TagName & ::= & \text{SendTo} \mid \text{DeptOnly} \mid \ldots \\
Domain & ::= & \_ \mid Subset \mid Subset\ Domain \\
SubSet & ::= & \{Subject\} \mid Subset \cup \{Subject\} \\
Subject & ::= & s_1 \mid \ldots \mid s_m
\end{array}$$

Here "*Caveat Cavts*" is the resulted sequence of putting $Caveat$ to the front of the sequence $Cavts$ as the first element of the new sequence, and the same explanation should also be made for "*Subset Domain*".

5.2 Basic Mechanisms

Intuitively, within a security labelling system, the labelling function accepts an object as input, and outputs a security label, which will be bound to this object. In order to implement the system, one needs to build a number of mechanisms to perform a variety of functions. In our model, the security labelling function is performed by three major mechanisms. They are:

- PIM (Policy Identification Mechanism): a mechanism to determine which policy is applied to a specific object;
- SLFM (Security Level Finding Mechanism): a mechanism to find the security level for any given object to be labelled;
- CCM (Caveat Choosing Mechanism): a mechanism to determine what caveats may or should be chosen for constructing the label for an object;

All these mechanisms operate based on the security labelling framework, which consists of three modules, SOD (Subjects & Objects Database), PM (Policy Management), and TR (Tags Registration).

Other two important components contained in a labelling system are:

- VRE (Validation & Reasoning Engine): a reasoning engine used to check the consistency of security labels produced. It directly connects with the lebelling framework as well as the security labelling function.
- LBM (Label Binding Mechanism): a mechanism applied to bind a valid label to an object.

Figure 2 shows all these mechanisms and modules, which form main components of a labelling system, and their connections. Note that in most systems PIM, SLFM, and CCM require human input and judgement.

5.3 Binding a Security Label to an Object

For binding the security label to an information object, we may or may not use a cryptographic service (ITU-T Recommendation X.841 2000). The methods are as follows:

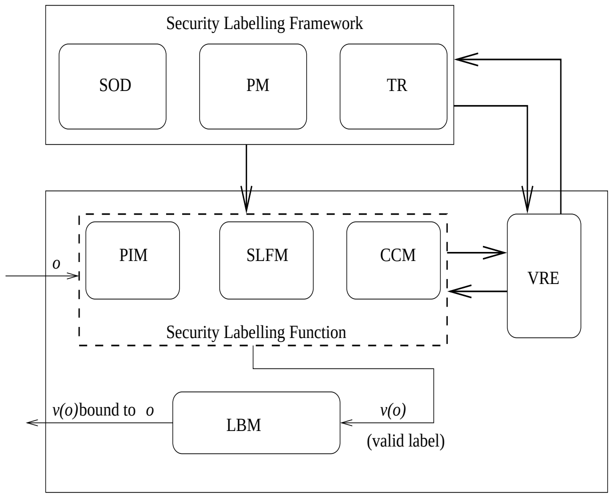


Figure 2: Main components of a labelling system

- *A method without the use of a cryptographic service.* In this method, a copy of the data (object o) and a copy of the security label ($\nu(o)$) are stored together, as a data record, inside the secure boundary of the system. In the case, we assume that the system is capable of protecting the integrity of the security label and the integrity, as well as possibly the secrecy, of the data. With this binding method, no cryptographic function is needed for the binding.
- *A method with the use of a cryptographic service.* Using a digital signature algorithm ($SigAlg$) and the private key (K_s) of a public key algorithm, a digital signature $SigAlg(K_s, H(o), \nu(o))$ is generated, where H is a public function such that $H(o)$ does not reveal information about o. In the case, the digital signature $SigAlg(K_s, H(o), \nu(o))$ is stored together with o and $\nu(o)$ in a data record; the generated digital signature binds $\nu(o)$ to o. With this binding method, $\nu(o)$ and $SigAlg(K_s, H(o), \nu(o))$ need not be stored inside the secure boundary of the system. If a cryptographic service is invoked with an incorrect value of $\nu(o)$, o or $SigAlg(K_s, H(o), \nu(o))$, the inconsistency is detected. This is accomplished using the public key of the public key algorithm as a verification key to verify the signature.

5.4 Label Validation

Security labels are generated by the labelling system based on the syntax above and intended as an extension to end system labels. It is necessary to ensure the integrity of the labels and their binding to the corresponding objects. That is, it is important to check whether the label marked on an object is valid before the label is to be used for security purpose.

Given a model $\mathbf{M} = (\langle \mathcal{S}, \mathcal{O}, (\mathcal{L}, \leq), \mathcal{C}, \mathcal{P}, F \rangle, \nu)$, let $\nu(o) = [P, L, X_1 \ldots X_k]$. Within the label validation procedure, the major task is to check whether the label $\nu(o)$ violates the policy P. We employ the following algorithm to check the validity of $\nu(o)$:

1. If there exists $s \in Scope_L(X_1 \ldots X_k)$ such that $f_l(s) < L$, then the checking process terminates and outputs $\nu(o)$ is invalid; otherwise,
2. If there exists $s \in Scope_L(X_1 \ldots X_k)$ and $\neg can\_have(s, o)$ is derived based on $SD(P, o)$, then the checking process terminates and outputs $\nu(o)$ is invalid; otherwise,
3. If all elements in $Scope_L(X_1 \ldots X_k)$ have been checked and the checking process does not terminate at step 1 or step 2, then the process terminates and outputs $\nu(o)$ is valid.

The policy identified to be used for making a label must not be violated by the label itself. In the verification procedure, it is important to check whether the policy is satisfied. In Step 1 of our algorithm, a very simple condition is used to assess the validity status of a label. Note that $L = f_l(o)$ in our model, the condition can be stated as follows:

- The security level of those subjects, to which an object labelled is intended to be delivered, can not be less than the security level of the object.

Similarly to this condition, the formula $f_l(o) \leq f_l(s)$ has been used for mandatory access control systems (MAC), see (Spalka, Cremers & Lehmler 2000).

Step 2 is to check whether there is any conflict between the security domain $SD(P, o)$ and the touch domain $TD(\nu(o), o)$. The intuitive meaning of the condition employed in this step is:

- If o is possibly touched by s but $\neg\ can\_have(s, o)$ can be derived from the fact that $SD(P, o)$ is the security domain related to P and o, then there is a conflict, which leads to that $\nu(o)$ is invalid.

For example, if P is the policy represented as follows:

$$(\forall e : H)\ be\_secret(e, o),$$

which means that o must be kept secret from all subjects in H. Therefore, we must have

$$(\forall e : H)\ (e \notin Scope(X_1 \ldots X_k)).$$

6 Discussion and Security Considerations

With security labelling, some problems have arisen through practical applications. We discuss some of these sorts of problems below, which need to be carefully considered in the design and implementation of a labelling system.

6.1 Threats

Security labels are implemented within both government agencies and commercial entities, to protect national and commercial information, respectively. This section is not intended to be a threat analysis, rather it highlights the distinction between threats at security levels and between classification systems. It is important to note that different threats affect different types of information. In order to protect information adequately Schneier (Schneier 2000) notes the importance of understanding the real threats to the system, and ensuring that the countermeasures implemented protect against and solve the right threats.

Government agencies recognise foreign intelligence services, terrorist organisations, and disgruntled individuals as potential sources of threats. This is not an exhaustive list, merely an example of possible threats. For example adversary nation states may have available to them a large resource pool, in both a financial and capability sense. In conjunction with the capabilities to perform an attack, there would also be an interest in the information providing a motivation. A possible attack from such a threat would be that of communication interception with a cryptanalysis attack. As an attack of this nature constitutes a high threat the strength of countermeasure and the mechanisms involved would be required to be strong.

In contrast commercial information should primarily be protected from disgruntled groups or individuals, along with interest in proprietary information from both commercial competitors and criminal groups. However the threat from foreign intelligence services has not been excluded, merely deemphasised. The threats that would be of principal concern would not generally come in the form of cryptanalysis, therefore allowing less stringent encryption methods to be employed. In this circumstance attacks from within the organisation or agency, would be considered to pose a significant threat. An example of this may include internal users attempting to gain unauthorised access to data. The findings of the 2002 Australian Computer Crime and Security Survey (AusCert 2002) show that 58 percent of respondents identified disgruntled employees or contractors as the most likely source of attacks, preceded only by independent hackers (73%). This may even extend to an employee being bribed or selling information to an external source. Therefore the internal security policy would provide a degree of protection against such a threat.

For all organisations that have public connections to the Internet there is a significant threat from hackers attacking the system from the external points, web connections and organisational web pages. Hackers operating with independent motivations are difficult to predict, however a proactive approach is always the best, followed by quick responses to alleviate vulnerabilities.

6.2 Prioritised Access Control Models

One problem with previous approaches to access control policy approaches, is that they are based on an idealisation of the true problem. They provide a first approximation: may or may not a subject access a given object? This binary, logical function is the essential starting point, but is generally insufficient to guide the hard decisions that are required in implementation. To provide such guidance - or to provide evaluative criteria by which adequacy of security architectures may be assessed and compared - a fine-grain statement of the full access control requirements is required.

In the positive case — where access to the object is granted — there has been a work in refining this policy statement. Most of this refinement has been in the area of the Quality of Service (QoS). As such this does not directly refer to the security of the access, although QoS may include availability and integrity aspects.

It is however in the negative case — where access is denied — that there is the most need to provide additional elaboration to a security policy formulation. Although it is nice to imagine that whenever a security policy asks for access to be denied, that this can be easily enforced in a uniformly strong and secure way, in reality this can rarely be achieved. Instead, efficient use of a finite security budget means that prioritisation needs to be applied in the use of access control enforcement mechanisms. It can be argued that a large part of the value of a good security architecture is in the way in which it efficiently combines security and other elements to implement security policy with appropriate protections against the threats.

How is this prioritisation to be achieved, and how is it to be specified? It is clearly infeasible for even a moderate system to specify the required type and level of enforcement required between each (subject, object) pair for which access is denied. Naturally the concept of grouping subjects (and possibly objects) as generally used in access control policies, is a powerful mechanism that can be adopted here, but note that the groupings that are useful for positive statements granting access, are unlikely to be reusable for statements about access enforcement. For example, a corporation's ACDF may be based on its business units (production, finance, sales etc.) while the enforcement policy may group subjects by location type (head office, field agent, overseas station, etc.)

To determine prioritisations, in theory all the relevant risks and costs associated with managing security could be captured and combined to provide an integrated cost model, effectively reducing the problem to a standard risk/economic optimisation problem. Note that with security this is considerably more complex than with more traditional risk management areas, since the threats are responsive and predictive: it enters the area of Games Theory (McCabe, Rassenti & Smith 1996). In any case, such a fine level of detail is again unmanageable and unachievable. Instead appropriate abstractions must be used.

In our approach we make some simplifying assumptions regarding how to derive the appropriate type and level of protection. We focus on two key aspects: the *value* to us of keeping the subject from accessing the object; and the *ability* of the subject to carry out an attack. The policy then states the strength to which we desire to prevent unauthorised accesses across the range of different attacks.

Consider a function

$$val : \mathcal{S} \times \mathcal{O} \to \mathcal{V}$$

where $\mathcal{V}$ represents the value to us of preventing the subject to access that object. For this paper, we consider Value as $\{Low, Med, High\}$, but may be represented by any scale of values that might be in use or understood by a given organisation (e.g. dollar value of intellectual property revealed, or the cost to insure against loss (Reiter& Stubblebine 1997)).

The second aspect we wish to formalise is the feasibility of a hostile subject carrying out different types of attacks. Such differences reflect many factors, including cost, time, technical ability, opportunity, risk aversion level, and strength of desire to gain access. We consider different types of attack to come from a space $\mathcal{A}$, and the feasibilities of attack from a set $\mathcal{F} = \{Easy, Achievable, Hard\}$. Again, more quantitative measures, such as probabilistic likelihood of successful attack per year, could be used for attack feasibilities. For each pair (s, a) we define the feasibility of subject S carrying out attack a:

$$feas : \mathcal{S} \times \mathcal{A} \to \mathcal{F}$$

We can now begin our model of a refined security policy. Firstly we need to decide what value objects must be protected against what feasibility of attacks - this defines the policy. Define a predicate $Pol \subseteq \mathcal{V} \times \mathcal{F}$, such that $Pol(v, f)$ holds whenever we require that all attacks of feasibility f against objects of value v are prevented. So, given an implementation security architecture that defends object o from attack a by subject s whenever $Arch(s, o, a)$ holds, we say that this architecture satisfies the policy if:

$$\forall s, o, a : Pol(val(s, o), feas(s, a)) \to Arch(s, o, a)$$

6.2.1 Example Policy

We now consider a simplified and mythical version of the security policy implemented for a Government. For our simple example, we can describe two hostile subjects: *MsSpy* and *MrFraud*, representing the subject "user" communities of potentially hostile nation-states, and white-collar crime, respectively.

We also consider two different types of attack: it CryptAnalysis, and *BribeInsider*. The first relies on collecting communications, and decrypting it if required. The second attack involves gaining the confidence of a legitimate subject for accessing the desired object, who provides a copy. We now look at estimated feasibilities of attacks:

$$feas(MsSpy, CryptAnalysis) = Achievable$$
$$feas(MsSpy, BribeInsider) = Achievable$$
$$feas(MrFraud, CryptAnalysis) = Hard$$
$$feas(MrFraud, BribeInsider) = Easy$$

Let us also consider the value of various documents. Suppose we have two documents (objects): *FighterContract*, and *FighterSpecs*. If *MrFraud* obtained a copy of the draft contract, then severe losses to the contractor, Defence, and the taxpayer will result. If *MsSpy* obtains the fighter technical specification, national security may be endangered. For this paper, we consider both these outcomes equally detrimental. Values of keeping documents from various subjects can thus be represented as follows:

$$val(MsSpy, FighterSpecs) = High$$
$$val(MsSpy, FighterContract) = Low$$
$$val(MrFraud, FighterContract) = High$$

Now the security policy can be represented by a relation in which $Pol(High, Achievable)$ holds (so achievable and easy attacks against $High$ value objects must be prevented). We can then deduce that for an architecture to meet this policy it must prevent $MsSpy$'s $CryptAnalysis$ or $BribeInsider$ attacks against $FighterSpecs$, and $MrFraud$'s $BribeInsider$ attack against $FigherContract$. Note that unless the policy is more restrictive than yet stated, it does not need to protect against $CryptAnalysis$ attacks against $FighterContract$, since $MrFraud$ does not find such attack's very feasible, while there is low value to keep that document from $MsSpy$.

7 Conclusion

A mechanically checkable model has been proposed for security labelling in an open system. The model supports the design and implementation of a security labelling system. A basis for label validation was also considered, however we did not provide details, which would be included in future work.

Future work may include an extended discussion on the dynamic management of security policies and its implementation. The aim would be to discuss the combination of security labels and associated security mechanisms to achieve the required security goals. Further analysis of threats and attacks that directly affect security labels and labelling systems may also be considered.

Acknowledgements

Many thanks to Dr. Maris Ozols for helpful discussions, valuable comments and suggestions. The work presented in this article has been supported in part by an Australian Research Council (ARC) Discovery Project grant.

References

FIPS PUB 188. (1994), *Standard Security Label for Information Transfer.* Available from www.itl.nist.gov/fipspubs/fip188.htm.

ISO/IEC 7498-1 (1994), *Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model.* ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission).

AusCert, Deloitte Touche Tohmatsu. (2002), Australian Computer Crime and Security Survey. http://www.auscert.org.au/Information/Auscert_info/2002cs.pdf.

Bell, D. E. & LaPadula, L. J. (1976), Secure Computer System: Unified exposition and Multics interpretation. MTR-2997, MITRE, Bedford, MA.

Biba, K. J. (1977), Integrity Consideration for Secure Computer Systems. MTR-3153, The Mitre Corporation.

Internet CIPSO Working Group. (1993), *Common IP Security Option Version 2.3.* Internet Draft.

Heintze, N. & Riecke, J. G. (1998), The slam calculus: Programming with secrecy and integrity. In *Proceedings of 25th ACM Symposium on Principles of Programming Languages (POPL)*, pages 365–377, San Diego, California.

Housley, R. (1993), *Security Labeling Framework for the Intenet.* Internet RFC 1457, May 1993.

ITU-T Recommendation X.841 (2000), *Information technology - Security Techniques - Security information objects for access control.*

McCabe, K., Rassenti, S. & Smith, V. (1996), Game theory and reciprocity in some extensive form experimental games. *Proceeding of The National Academy of Science*, 93:13421–13428.

Myers, A. C. & Liskov, B. (2000), Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology*, 9(4):410–442.

Reiter, M.K. & Stubblebine, S.G.(1997), Toward acceptable metrics of authentication. In *Proceedings of the IEEE Symposium on Security and Privacy.*

Schneier, B.(2000), *Secrets and Lies – Digital Security in a Networked World.* John Wiley & Sons.

Spalka, A., Cremers, A.B. & Lehmler, H. (2000), Protecting confidentiality against trojan horse programs in discretionary access control system. In *Proceedings of the 5th Australasian Conference on Information Security and Privacy (ACISP 2000)*, volume 1841 of *Lecture Notes in Computer Science*, pages 1–17. Springer.

Zdancewic, S. Zheng, L., Nystrom, N. & Myers, A.C. (2001), Untrusted hosts and confidentiality: Secure program partitioning. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, pages 1–14, Banff, Canada.

Zheng, L. & Myers, A.C. (2004), Dynamic security labels and noninterference. In *Proceedings of the 2nd International Workshop on Formal Aspects in Security and Trust (FAST)*, Toulouse, France.

Improvements of TLAESA Nearest Neighbour Search Algorithm and Extension to Approximation Search

Ken Tokoro **Kazuaki Yamaguchi** **Sumio Masuda**

Kobe University,
1-1, Rokkodai, Nada-ku, Kobe 657-8501 Japan,
Email: ky@kobe-u.ac.jp

Abstract

Nearest neighbour (NN) searches and k nearest neighbour (k-NN) searches are widely used in pattern recognition and image retrieval. An NN (k-NN) search finds the closest object (closest k objects) to a query object. Although the definition of the distance between objects depends on applications, its computation is generally complicated and time-consuming. It is therefore important to reduce the number of distance computations. TLAESA (Tree Linear Approximating and Eliminating Search Algorithm) is one of the fastest algorithms for NN searches. This method reduces distance computations by using a branch and bound algorithm. In this paper we improve both the data structure and the search algorithm of TLAESA. The proposed method greatly reduces the number of distance computations. Moreover, we extend the improved method to an approximation search algorithm which ensures the quality of solutions. Experimental results show that the proposed method is efficient and finds an approximate solution with a very low error rate.

Keywords: Nearest Neighbour Search, k Nearest Neighbour Search, TLAESA, Approximation Search, Distance Computaion.

1 Introduction

NN and k-NN searches are techniques which find the closest object (closest k objects) to a query object from a database. These are widely used in pattern recognition and image retrieval. We can see examples of their applications to handwritten character recognition in (Rico-Juan & Micó 2003) and (Micó & Oncina 1998), and so on. In this paper we consider NN (k-NN) algorithms that can work in any metric space. For any x, y, z in a metric space, the distance function $d(\cdot,\cdot)$ satisfies the following properties:

$$d(x,y) = 0 \Leftrightarrow x = y,$$
$$d(x,y) = d(y,x),$$
$$d(x,z) \leq d(x,y) + d(y,z).$$

Although the definition of the distance depends on applications, its calculation is generally complicated and time-consuming. We particularly call the calculation of $d(\cdot,\cdot)$ a *distance computation*.

For the NN and k-NN searches in metric spaces, some methods that can manage a large set of objects efficiently have been introduced(Hjaltason & Samet 2003). They are categorized into two groups. The methods in the first group manage objects with a tree structure such as vp-tree(Yianilos 1993), M-tree(Ciaccia, Patella & Zezula 1997), sa-tree (Navarro 2002) and so forth. The methods in the second group manage objects with a distance matrix, which stores the distances between objects. The difference between two groups is caused by their approaches to fast searching. The former aims at reducing the computational tasks in the search process by managing objects effectively. The latter works toward reducing the number of distance computations because generally their costs are higher than the costs of other calculations. In this paper we consider the latter approach.

AESA (Approximating and Eliminating Search Algorithm)(Vidal 1986) is one of the fastest algorithms for NN searches in the distance matrix group. The number of distance computations is bounded by a constant, but the space complexity is quadratic. LAESA (Linear AESA)(Micó, Oncina & Vidal 1994) was introduced in order to reduce this large space complexity. Its space complexity is linear and its search performance is almost the same as that of AESA. Although LAESA is more practical than AESA, it is impractical for a large database because calculations other than distance computations increase. TLAESA (Tree LAESA)(Micó, Oncina & Carrasco 1996) is an improvement of LAESA and reduces the time complexity to sublinear. It uses two kinds of data structures: a distance matrix and a binary tree, called a search tree.

In this paper, we propose some improvements of the search algorithm and the data structures of TLAESA in order to reduce the number of distance computations. The search algorithm follows the best first algorithm. The search tree is transformed to a multiway tree from a binary tree. We also improve the selection method of the root object in the search tree. These improvements are simple but very effective. We then introduce the way to perform a k-NN search in the improved TLAESA. Moreover, we propose an extension to an approximation search algorithm that can ensure the quality of solutions.

This paper is organized as follows. In section 2, we describe the details of the search algorithm and the data structures of TLAESA. In section 3, we propose some improvements of TLAESA. In section 4, we present an extension to an approximation search algorithm. In section 5, we show some experimental results. Finally, in section 6, we conclude this paper.

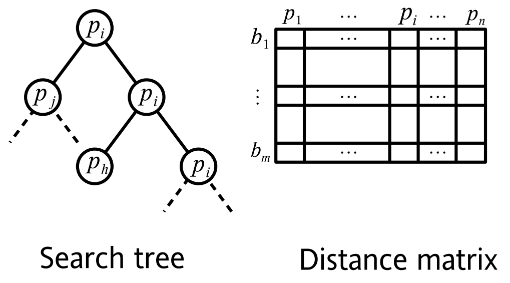


Figure 1: An example of the data structures in TLAESA.

2 TLAESA

TLAESA uses two kinds of data structures: the distance matrix and the search tree. The distance matrix stores the distances from each object to some selected objects. The search tree manages hierarchically all objects. During the execution of the search algorithm, the search tree is traversed and the distance matrix is used to avoid exploring some branches.

2.1 Data Structures

We explain the data structures in TLAESA. Let P be the set of all objects and B be a subset consisting of selected objects called *base prototypes*. The distance matrix M is a two-dimensional array that stores the distances between all objects and base prototypes. The search tree T is a binary tree such that each node t corresponds to a subset $S_t \subset P$. Each node t has a pointer to the representative object $p_t \in S_t$ which is called a *pivot*, a pointer to a left child node l, a pointer to a right child node r and a covering radius r_t. The covering radius is defined as

$$r_t = \max_{p \in S_t} d(p, p_t). \tag{1}$$

The pivot p_r of r is defined as $p_r = p_t$. On the other hand, the pivot p_l of l is determined so that

$$p_l = \operatorname*{argmax}_{p \in S_t} d(p, p_t). \tag{2}$$

Hence, we have the following equality:

$$r_t = d(p_t, p_l). \tag{3}$$

S_t is partitioned into two disjoint subsets S_r and S_l as follows:

$$\begin{aligned} S_r &= \{p \in S_t | d(p, p_r) < d(p, p_l)\}, \\ S_l &= S_t - S_r. \end{aligned} \tag{4}$$

Note that if t is a leaf node, $S_t = \{p_t\}$ and $r_t = 0$. Fig. 1 shows an example of the data structures.

2.2 Construction of the Data Structures

We first explain the construction process of the search tree T. The pivot p_t of the root node t is randomly selected and S_t is set to P. The pivot p_l of the left child node and the covering radius r_t are defined by Eqs. (2) and (3). The pivot p_r of the right child node is set to p_t. S_t is partitioned into S_r and S_l by Eq. (4). These operations are recursively repeated until $|S_t| = 1$.

The distance matrix M is constructed by selecting base prototypes. This selection is important because base prototypes are representative objects which are used to avoid some explorations of the tree.

The ideal selection of them is that each object is as far away as possible from other objects. In (Micó et al. 1994), a greedy algorithm is proposed for this selection. This algorithm chooses an object that maximizes the sum of distances from the other base prototypes which have already been selected. In (Micó & Oncina 1998), another algorithm is proposed, which chooses an object that maximizes the minimum distance to the preselected base prototypes. (Micó & Oncina 1998) shows that the latter algorithm is more effective than the former one. Thus, we use the later algorithm for the selection of base prototypes.

The search efficiency depends not only on the selection of base prototypes but also on the number of them. There is a trade-off between the search efficiency and the size of distance matrix, i.e. the memory capacity. The experimental results in (Micó et al. 1994) show that the optimal number of base prototypes depends on the dimensionality dm of the space. For example, the optimal numbers are 3, 16 and 24 if $dm = 2, 4$ and 8, respectively. The experimental results also show that the optimal number does not depend on the number of objects.

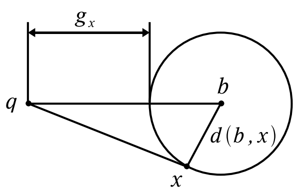


Figure 2: Lower bound.

2.3 Search Algorithm

The search algorithm follows the branch and bound strategy. It traverses the search tree T in the depth first order. The distance matrix M is referred whenever each node is visited in order to avoid unnecessary traverse of the tree T. The distance are computed only when a leaf node is reached.

Given a query object q, the distance between q and the base prototypes are computed. These results are stored in an array D. The object which is the closest to q in B is selected as the nearest neighbour candidate p_{min}, and the distance $d(q, p_{min})$ is recorded as d_{min}. Then, the traversal of the search tree T starts at the root node. The lower bound for the left child node l is calculated whenever each node t is reached if it is not a leaf node. The lower bound of the distance between q and an object x is defined as

$$g_x = \max_{b \in B} |d(q, b) - d(b, x)|. \tag{5}$$

See Fig. 2. Recall that $d(q, b)$ was precomputed before the traversals and was stored in D. In addition, the value $d(b, x)$ was also computed during the construction process and stored in the distance matrix M. Therefore, g_x is calculated without any actual distance computations. The lower bound g_x is not actual distance $d(q, x)$. Thus, it does not ensure that the number of visited nodes in the search becomes minimum. Though, this evaluation hardly costs, hence it is possible to search fast. The search process accesses the left child node l if $g_{p_l} \le g_{p_r}$, or the right child node r if $g_{p_l} > g_{p_r}$. When a leaf node is reached, the distance is computed and both p_{min} and d_{min} are updated if the distance is less than d_{min}.

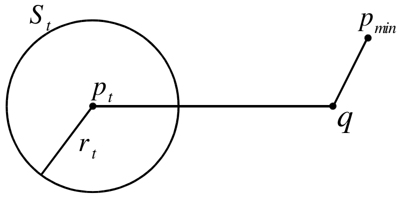


Figure 3: Pruning Process.

procedure NN search(q)

```
1: $t \leftarrow$ root of $T$
2: $d_{min} = \infty, g_{p_t} = 0$
3: for $b \in B$ do
4:     $D[b] = d(q, b)$
5:     if $D[b] < d_{min}$ then
6:         $p_{min} = b, d_{min} = D[b]$
7:     end if
8: end for
9: $g_{p_t} = \max_{b \in B} |(D[b] - M[b, p_t])|$
10: search($t, g_{p_t}, q, p_{min}, d_{min}$)
11: return $p_{min}$
```

Figure 4: Algorithm for an NN search in TLAESA.

We explain the pruning process. Fig. 3 shows the pruning situation. Let t be the current node. If the inequality

$$d_{min} + r_t < d(q, p_t) \tag{6}$$

is satisfied, we can see that no object exists in S_t which is closer to q than p_{min} and the traversal to node t is not necessary. Since $g_{p_t} \leq d(q, p_t)$, Eq. (6) can be replaced with

$$d_{min} + r_t < g_{p_t}. \tag{7}$$

Figs. 4 and 5 show the details of the search algorithm(Micó et al. 1996).

3 Improvements of TLAESA

In this section, we propose some improvements of TLAESA in order to reduce the number of distance computations.

3.1 Tree Structure and Search Algorithm

If we can evaluate the lower bounds g in the ascending order of their values, the search algorithm runs very fast. However, this is not guaranteed in TLAESA since the evaluation order is decided according to the tree structure. We show such an example in Fig. 6. In this figure, u, v and w are nodes. If $g_{p_v} < g_{p_w}$, it is desirable that v is evaluated before w. But, if $g_{p_v} > g_{p_u}$, w might be evaluated before v.

We propose the use of a multiway tree and the best first order search instead of a binary tree and the depth first search. During the best first search process, we can traverse preferentially a node whose subset may contain the closest object. Moreover, we can evaluate more nodes at one time by using of the multiway tree. The search tree in TLAESA has many nodes which have a pointer to the same object. In the proposed structure, we treat such nodes as one node. Each node t corresponds to a subset $S_t \subset P$ and has a pivot p_t, a covering radius $r_t = \max_{p \in S_t} d(p, p_t)$ and pointers to its children nodes.

procedure search($t, g_{p_t}, q, p_{min}, d_{min}$)

```
1: if $t$ is a leaf then
2:     if $g_{p_t} < d_{min}$ then
3:         $d = d(q, p_t)$ {distance computation}
4:         if $d < d_{min}$ then
5:             $p_{min} = p_t, d_{min} = d$
6:         end if
7:     end if
8: else
9:     $r$ is a right child of $t$
10:    $l$ is a left child of $t$
11:    $g_{p_r} = g_{p_t}$
12:    $g_{p_l} = \max_{b \in B} |(D[b] - M[b, p_t])|$
13:    if $g_{p_l} < g_{p_r}$ then
14:        if $d_{min} + r_l > g_{p_l}$ then
15:            search($l, g_{p_l}, p_{min}, d_{min}$)
16:        end if
17:        if $d_{min} + r_r > g_{p_r}$ then
18:            search($r, g_{p_r}, p_{min}, d_{min}$)
19:        end if
20:    else
21:        if $d_{min} + r_r > g_{p_r}$ then
22:            search($r, g_{p_r}, p_{min}, d_{min}$)
23:        end if
24:        if $d_{min} + r_l > g_{p_l}$ then
25:            search($l, g_{p_l}, p_{min}, d_{min}$)
26:        end if
27:    end if
28: end if
```

Figure 5: A recursive procedure for an NN search in TLAESA.

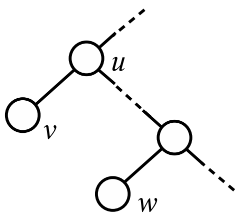


Figure 6: A case in which the search algorithm in TLAESA does not work well.

We show a method to construct the tree structure in Fig. 7. We first select randomly the pivot p_t of the root node t and set S_t to P. Then we execute the procedure **makeTree**(t, p_t, S_t) in Fig. 7.

We explain the search process in the proposed structure. The proposed method maintains a priority queue Q that stores triples (node t, lower bound g_{p_t}, covering radius r_t) in the increasing order of $g_{p_t} - r_t$. Given a query object q, we calculate the distances between q and base prototypes and store their values in D. Then the search process starts at the root of T. The following steps are recursively repeated until Q becomes empty. When t is a leaf node, the distance $d(q, p_t)$ is computed if $g_{p_t} < d_{min}$. If t is not a leaf node and its each child node t' satisfies the inequality

$$g_{p_t} < r_{t'} + d_{min}, \tag{8}$$

the lower bound $g_{p_{t'}}$ is calculated and a triple $(t', g_{p_{t'}}, r_{t'})$ is added to Q. Figs. 8 and 9 show the details of the algorithm.

procedure makeTree(t, p_t, S_t)

```
1:  $t' \leftarrow$ new child node of $t$
2:  if $|S_t| = 1$ then
3:      $p_{t'} = p_t$ and $S_{t'} = \{p_{t'}\}$
4:  else
5:      $p_{t'} = \operatorname{argmax}_{p \in S_t} d(p, p_t)$
6:      $S_{t'} = \{p \in S_t | d(p, p_{t'}) < d(p, p_t)\}$
7:      $S_t = S_t - S_{t'}$
8:      makeTree($t', p_{t'}, S_{t'}$)
9:      makeTree($t, p_t, S_t$)
10: end if
```

Figure 7: Method to construct the proposed tree structure.

procedure NN search(q)

```
1:  $t \leftarrow$ root of $T$
2:  $d_{min} = \infty, g_{p_t} = 0$
3:  for $b \in B$ do
4:      $D[b] = d(q, b)$
5:      if $D[b] < d_{min}$ then
6:          $p_{min} = b, d_{min} = D[b]$
7:      end if
8:  end for
9:  $g_t = \max_{b \in B} |(D[b] - M[b, p_t])|$
10: $Q \leftarrow \{(t, g_{p_t}, r_t)\}$
11: while $Q$ is not empty do do
12:     $(t, g_{p_t}, r_t) \leftarrow$ element in $Q$
13:     search($t, g_{p_t}, q, p_{min}, d_{min}$)
14: end while
15: return $p_{min}$
```

Figure 8: Proposed algorithm for an NN search.

procedure search($t, g_{p_t}, q, p_{min}, d_{min}$)

```
1:  if $t$ is a leaf then
2:      if $g_{p_t} < d_{min}$ then
3:          $d = d(q, p_t)$ {distance computation}
4:          if $d < d_{min}$ then
5:              $p_{min} = p_t, d_{min} = d$
6:          end if
7:      end if
8:  else
9:      for each child $t'$ of $t$ do
10:         if $g_{p_t} < r_{t'} + d_{min}$ then
11:             $g_{p_{t'}} = \max_{b \in B} |(D[b] - M[b, p_{t'}])|$
12:             $Q \leftarrow Q \cup \{(t', g_{p_{t'}}, r_{t'})\}$
13:         end if
14:     end for
15: end if
```

Figure 9: A procedure used in the proposed algorithm for an NN search.

procedure k-NN search(q, k)

```
1:  $t \leftarrow$ root of $T$
2:  $d_{min} = \infty, g_{p_t} = 0$
3:  for $b \in B$ do
4:      $D[b] = d(q, b)$
5:      if $D[b] < d_{min}$ then
6:          $V \leftarrow V \cup \{(b, D[b])\}$
7:          if $|V| = k + 1$ then
8:              remove $(k + 1)$th pair from $V$
9:          end if
10:         if $|V| = k$ then
11:             $(c, d(q, c)) \leftarrow k$th pair of $V$
12:             $d_{min} = d(q, c)$
13:         end if
14:     end if
15: end for
16: $g_{p_t} = \max_{b \in B} |(D[b] - M[b, p_t])|$
17: $Q \leftarrow \{(t, g_{p_t}, r_t)\}$
18: while $Q$ is not empty do
19:     $(t, g_{p_t}, r_t) \leftarrow$ element in $Q$
20:     search($t, g_{p_t}, q, V, d_{min}, k$)
21: end while
22: return $k$ objects $\leftarrow V$
```

Figure 10: Proposed algorithm for a k-NN search.

3.2 Selection of Root Object

We focus on base prototypes in order to reduce node accesses. The lower bound of the distance between a query q and a base prototype b is

$$\begin{aligned} g_b &= \max_{b \in B} |d(q, b) - d(b, b)| \\ &= d(q, b). \end{aligned}$$

This value is not an estimated distance but an actual distance.

If we can use an actual distance in the search process, we can evaluate more effectively which nodes are close to q. This fact means that the search is efficiently performed if many base prototypes are visited in the early stage. In other words, it is desirable that more base prototypes are arranged in the upper part of the search tree. Thus, in the proposed algorithm, we choose the first base prototype b_1 as the root object.

3.3 Extension to a k-NN Search

LAESA was developed to perform NN searches and (Moreno-Seco, Micó & Oncina 2002) extended it so that k-NN searches can be executed. In this section, we extend the improved TLAESA to a k-NN search algorithm. The extension is simple modifications of the algorithm described above. We use a priority queue V for storing k nearest neighbour candidates and modify the definition of d_{min}. V stores pairs (object p, distance $d(q, p)$) in the increasing order of $d(q, p)$. d_{min} is defined as

$$d_{min} = \begin{cases} \infty & (|V| < k) \\ d(q, c) & (|V| = k) \end{cases} \tag{9}$$

where c is the object of the kth pair in V.

We show in Figs. 10 and 11 the details of the k-NN search algorithm. The search strategy essentially follows the algorithm in Figs. 8 and 9, but the k-NN search algorithm uses V instead of p_{min}.

(Moreno-Seco et al. 2002) shows that the optimal number of base prototypes depends on not only the dimensionality of the space but also the value of k and that the number of distance computations increases as k increases.

4 Extension to an Approximation Search

In this section, we propose an extension to an approximation k-NN search algorithm which ensures the

procedure search$(t, g_{p_t}, q, V, d_{min}, k)$

```
 1: if $t$ is a leaf then
 2:   if $g_{p_t} < d_{min}$ then
 3:     $d = d(q, p_t)$ {distance computation}
 4:     if $d < d_{min}$ then
 5:       $V \leftarrow V \cup \{(p_t, d(q, p_t))\}$
 6:       if $|V| = k + 1$ then
 7:         remove $(k+1)$th pair from $V$
 8:       end if
 9:       if $|V| = k$ then
10:         $(c, d(q, c)) \leftarrow k$th pair of $V$
11:         $d_{min} = d(q, c)$
12:       end if
13:     end if
14:   end if
15: else
16:   for each child $t'$ of $t$ do
17:     if $g_{p_t} < r_{t'} + d_{min}$ then
18:       $g_{p_{t'}} = \max_{b \in B} |(D[b] - M[b, p_{t'}])|$
19:       $Q \leftarrow Q \cup \{(t', g_{p_{t'}}, r_{t'})\}$
20:     end if
21:   end for
22: end if
```

Figure 11: A procedure used in the proposed algorithm for a k-NN search.

quality of solutions. Consider the procedure in Fig. 11. We replace the 4th line with

$$\textbf{if } d < \alpha \cdot d_{min} \textbf{ then}$$

and the 17th line with

$$\textbf{if } g_t < r_{t'} + \alpha \cdot d_{min} \textbf{ then}$$

where α is real number such that $0 < \alpha \leq 1$. The pruning process gets more efficient as these conditions become tighter.

The proposed method ensures the quality of solutions. We can show the approximation ratio to an optimal solution using α. Let a be the nearest neighbour object and a' be the nearest neighbour candidate object. If our method misses a and give a' as the answer, the equation

$$g(q, a) \geq \alpha \cdot d(q, a') \qquad (10)$$

is satisfied. Then a will be eliminated from targeted objects. Since $g(q, a) \leq d(q, a)$, we can obtain the following equation:

$$d(q, a') \leq \frac{1}{\alpha} d(q, a). \qquad (11)$$

Thus, the approximate solution are suppressed by $\frac{1}{\alpha}$ times of the optimal solution.

5 Experiments

In this section we show some experimental results and discuss them. We tested on an artificial set of random points in the 8-dimensional euclidean space. We also used the euclidean distance as the distance function. We evaluated the number of distance computations and the number of accesses to the distance matrix in 1-NN and 10-NN searches.

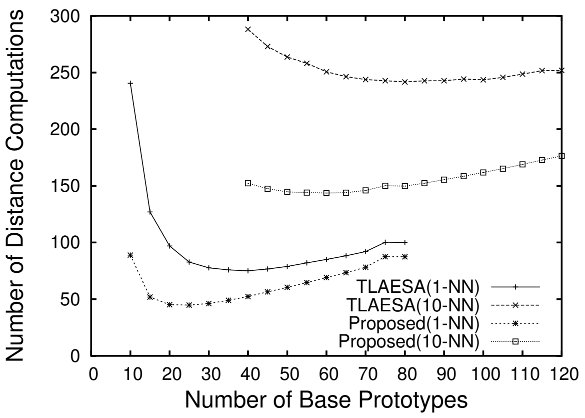


Figure 12: Relation of the number of distance computations to the number of base prototypes.

	1-NN	10-NN
TLAESA	40	80
Proposed	25	60

Table 1: The optimal number of base prototypes.

5.1 The Optimal Number of Base Prototypes

We first determined experimentally the optimal number of base prototypes. The number of objects was fixed to 10000. We executed 1-NN and 10-NN searches for various numbers of base prototypes, and counted the number of distance computations. Fig. 12 shows the results. From this figure, we chose the number of base prototypes as shown in Table. 1.

We can see that the values in the proposed method are fewer than those in TLAESA. This means that the proposed method can achieve better performance with smaller size of distance matrix. We used the values in Table. 1 in the following experiments.

5.2 Evaluation of Improvements

We tested the effects of our improvements described in 3.1 and 3.2. We counted the numbers of distance computations in 1-NN and 10-NN searches for various numbers of objects. The results are shown in Figs. 13 and 14. Similar to TLAESA, the number of the distance computations in the proposed method does not depend on the number of objects. In both of 1-NN and 10-NN searches, it is about 60% of the number of distance computations in TLAESA. Thus we can see that our improvements are very effective.

In the search algorithms of TLAESA and the proposed methods, various calculations are performed other than distance computations. The costs of the major part of such calculations are proportional to the number of accesses to the distance matrices. We therefore counted the numbers of accesses to the distance matrices. We examined the following two cases:

(i) TLAESA vs. TLAESA with the improvement of selection of the root object.

(ii) Proposed method only with improvement of tree structure and search algorithm vs. proposed method only with the improvement of selection of the root object.

In the case (i), the number of accesses to the distance matrix is reduced by 12% in 1-NN searches and 4.5% in 10-NN searches. In the case (ii), it is reduced by 6.8% in 1-NN searches and 2.7% in 10-NN searches.

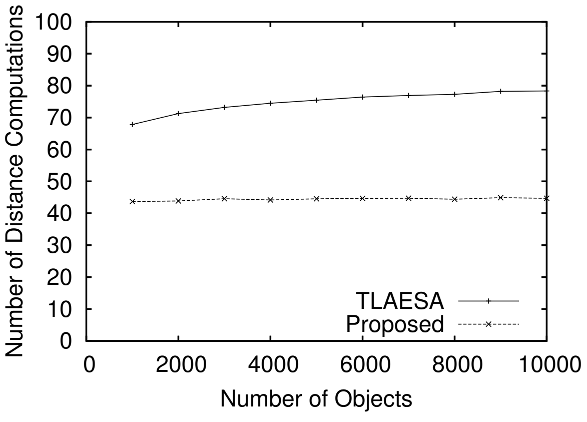


Figure 13: The number of distance computations in 1-NN searches.

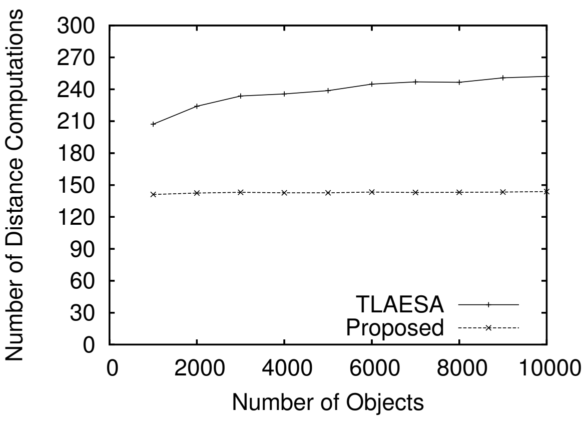


Figure 14: The number of distance computations in 10-NN searches.

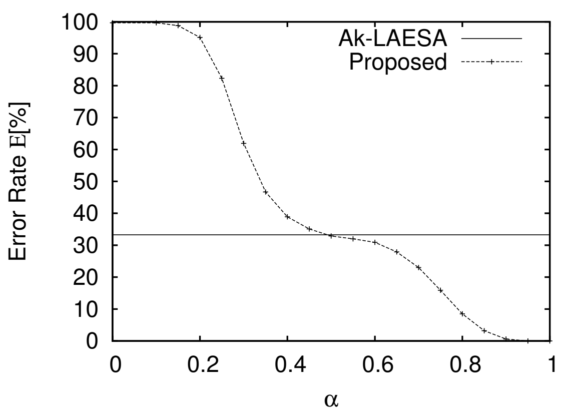


Figure 15: Error rate in 10-NN searches.

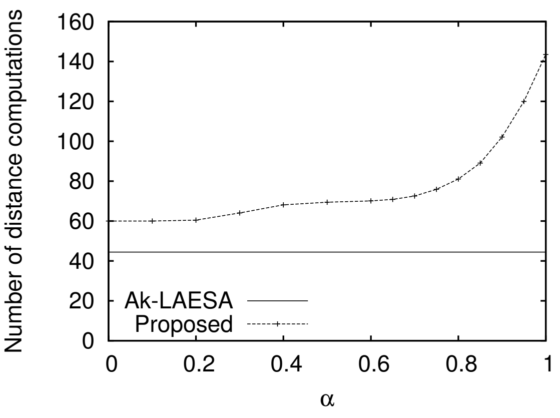


Figure 16: Relation of the number of distance computations to the value of α in 10-NN searches.

Thus we can see that this improvement about selection of the root object is effective.

5.3 Evaluation of Approximation Search

We tested the performance of the approximation search algorithm. We compared the proposed method to Ak-LAESA, which is the approximation search algorithm proposed in (Moreno-Seco, Micó & Oncina 2003). Each time a distance is computed in Ak-LAESA, the nearest neighbour candidate is updated and its value is stored. When the nearest neighbour object is found, the best k objects are chosen from the stored values. In Ak-LAESA, the number of distance computations of the k-NN search is exactly the same as that of the NN search.

To compare the proposed method with Ak-LAESA, we examined how many objects in the approximate solutions exist in the optimal solutions. Thus, we define the error rate E as follows:

$$E[\%] = \frac{|\{x_i | x_i \notin Opt, i = 1, 2, \cdots, k\}|}{k} \times 100 \quad (12)$$

where $\{x_1, x_2, \cdots, x_k\}$ is a set of k objects which are obtained by an approximation algorithm and Opt is a set of k closest objects to the query object.

Fig. 15 shows the error rate when the value of α is changed in 10-NN searches. Fig. 16 also shows the relation of the number of distance computations to the value of α in 10-NN searches. In the range $\alpha \geq 0.5$, the proposed method shows the lower error rate than Ak-LAESA. In particular, the error rate of the proposed method is almost 0 in range $\alpha \geq 0.9$. From two figures, we can control the error rate and the number of distance computations by changing the value of α. For example, the proposed method with $\alpha = 0.9$ reduces abount 28.6% of distance computations and its error rate is almost 0.

Then we examined the accuracy of the approximate solutions. We used $\alpha = 0.5$ for the proposed method because the error rate of the proposed method with $\alpha = 0.5$ is equal to the one of Ak-LAESA. We performed 10-NN searches 10000 times for each method and examined the distribution of kth approximate solution to kth optimal solution. We show the results in Figs. 17 and 18. In each figure, x axis represents the distance between a query object q and the kth object in the optimal solution. y axis shows the distance between q and the kth object in the approximate solution. The point near the line $y = x$ represents that kth approximate solution is very close to kth optimal solution. In Fig. 17, many points are widely distributed. In the worst case, some appriximate solutions reach about 3 times of the optimal solution. From these figures, we can see that the accuracy of solution by the proposed method is superior to the one by Ak-LAESA. We also show the result with $\alpha = 0.9$ in Fig. 19. Most points exist near the line $y = x$.

Though Ak-LAESA can reduce drastically the number of distance computations, its approximate solutions are often far from the optimal solutions. On the other hand, the proposed method can reduce the number of distance computations to some extent with

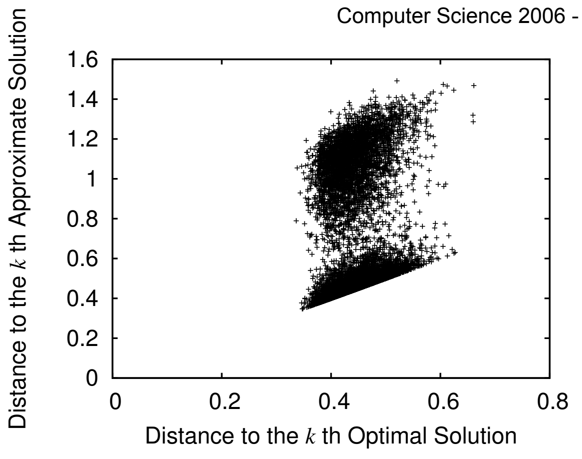


Figure 17: The distribution of the approximate solution by A*k*-LAESA to the optimal solution.

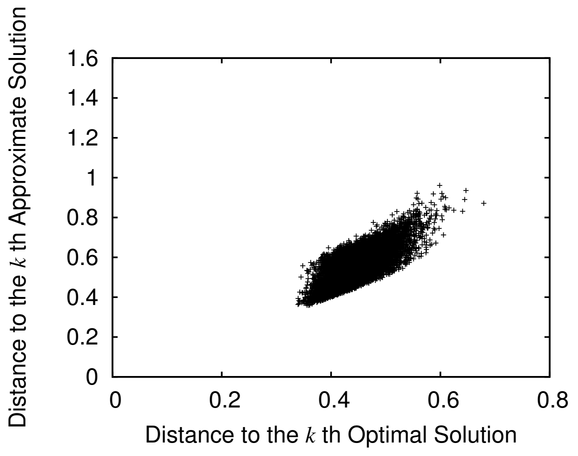


Figure 18: The distribution the approximate solution by the proposed method with $\alpha = 0.5$ to the optimal solution.

very low error rate. Moreover, the accuracy of its approximate solutions is superior to that of A*k*-LAESA.

6 Conclusions

In this paper, we proposed some improvements of TLAESA. In order to reduce the number of distance computations in TLAESA, we improved the search algorithm to best first order from depth first order and the tree structure to a multiway tree from a binary tree. In the 1-NN searches and 10-NN searches in a 8-dimensional space, the proposed method reduced about 40% of distance computations. We then proposed the selection method of root object in the search tree. This improvement is very simple but is effective to reduce the number of accesses to the distance matrix. Finally, we extended our method to an approximation *k*-NN search algorithm that can ensure the quality of solutions. The approximate solutions of the proposed method are suppressed by $\frac{1}{\alpha}$ times of the optimal solutions. Experimental results show that the proposed method can reduce the number of distance computations with very low error rate by selecting the appropriate value of α, and that the accuracy of the solutions is superior to A*k*-LAESA. From these viewpoints, the method presented in this paper is very effective when the distance computations are time-consuming.

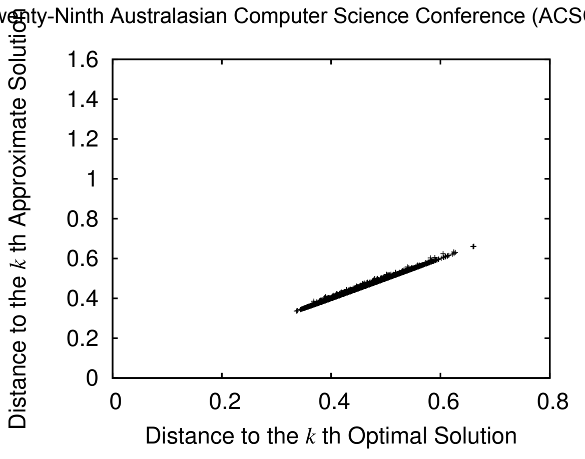


Figure 19: The distribution the approximate solution by the proposed method with $\alpha = 0.9$ to the optimal solution.

References

Ciaccia, P., Patella, M. & Zezula, P. (1997), M-tree: An efficient access method for similarity search in metric spaces, *in* 'Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)', pp. 426–435.

Hjaltason, G. R. & Samet, H. (2003), 'Index-driven similarity search in metric spaces', *ACM Transactions on Database Systems* **28**(4), 517–580.

Micó, L. & Oncina, J. (1998), 'Comparison of fast nearest neighbour classifiers for handwritten character recognition', *Pattern Recognition Letters* **19**(3-4), 351–356.

Micó, L., Oncina, J. & Carrasco, R. C. (1996), 'A fast branch & bound nearest neighbour classifier in metric spaces', *Pattern Recognition Letters* **17**(7), 731–739.

Micó, M. L., Oncina, J. & Vidal, E. (1994), 'A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements', *Pattern Recognition Letters* **15**(1), 9–17.

Moreno-Seco, F., Micó, L. & Oncina, J. (2002), 'Extending LAESA fast nearest neighbour algorithm to find the k-nearest neighbours', *Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence* **2396**, 691–699.

Moreno-Seco, F., Micó, L. & Oncina, J. (2003), 'A modification of the LAESA algorithm for approximated k-NN classification', *Pattern Recognition Letters* **24**(1-3), 47–53.

Navarro, G. (2002), 'Searching in metric spaces by spatial approximation', *The VLDB Journal* **11**(1), 28–46.

Rico-Juan, J. R. & Micó, L. (2003), 'Comparison of AESA and LAESA search algorithms using string and tree-edit-distances', *Pattern Recognition Letters* **24**(9-10), 1417–1426.

Vidal, E. (1986), 'An algorithm for finding nearest neighbours in (approximately) constant average time', *Pattern Recognition Letters* **4**(3), 145–157.

Yianilos, P. N. (1993), Data structures and algorithms for nearest neighbor search in general metric spaces, *in* 'SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms', pp. 311–321.

Trust Network Analysis with Subjective Logic

Audun Jøsang[1] **Ross Hayward**[1] **Simon Pope**[2]

[1]School of Software Engineering and Data Communications[*]
Queensland University of Technology, Brisbane, Australia
Email: {a.josang, r.hayward}@qut.edu.au

[2]CRC for Enterprise Distributed Systems Technology (DSTC Pty Ltd)[†]
The University of Queensland, Brisbane, Australia
Email: skjpope@gmail.com

Abstract

Trust networks consist of transitive trust relationships between people, organisations and software agents connected through a medium for communication and interaction. By formalising trust relationships, e.g. as reputation scores or as subjective trust measures, trust between parties within the community can be derived by analysing the trust paths linking the parties together. This article describes a method for trust network analysis using subjective logic (TNA-SL). It provides a simple notation for expressing transitive trust relationships, and defines a method for simplifying complex trust networks so that they can be expressed in a concise form and be computationally analysed. Trust measures are expressed as beliefs, and subjective logic is used to compute trust between arbitrary parties in the network. We show that TNA-SL is efficient, and illustrate possible applications with examples.

1 Introduction

Modern communication media are increasingly removing us from the familiar styles of interacting that traditionally rely on some degree of pre-established trust between business partners. Moreover, most traditional cues for assessing trust in the physical world are not available through those media. We may now be conducting business with people and organisations of which we know nothing, and we are faced with the difficult task of making decisions involving risk in such situations. As a result, the topic of trust in open computer networks is receiving considerable attention in the network security community and e-commerce industry [1, 4, 13, 18, 19, 23, 26]. State of the art technology for stimulating trust in e-commerce includes cryptographic security mechanisms for providing confidentiality of communication and authentication of identities. However, merely having a cryptographically certified identity or knowing that the communication channel is encrypted is not enough for making informed decisions if no other knowledge about a remote transaction partner is available. Trust therefore also applies to the truthfulness of specific claims made by parties who request services in a given business context as described in the WS-Trust specifications [26], and trust between business partners regarding security assertions as described in the Liberty Alliance Framework [18, 19]. Trust also applies to the honesty, reputation and reliability of service providers or transaction partners, in general or for a specific purpose. In this context, the process of assessing trust becomes part of quality of service (QoS) evaluation, decision making and risk analysis.

Being able to formally express and reason with these types of trust is needed not only to create substitutes for the methods we use in the physical world, like for instance trust based on experiences or trust in roles, but also for creating entirely new methods for determining trust that are better suited for computerised interactions. This will facilitate the creation of communication infrastructures where trust can thrive in order to ensure meaningful and mutually beneficial interactions between players.

The main contribution of this paper is a method for discovering trust networks between specific parties, and a practical method for deriving measures of trust from such networks. Our method, which is called TNA-SL (Trust Network Analysis with Subjective Logic), is based on analysing trust networks as directed series-parallel graphs that can be represented as canonical expressions, combined with measuring and computing trust using subjective logic. We finally provide a numerical example of how trust can be analysed and computed using our method.

2 Trust Transitivity

Trust transitivity means, for example, that if Alice trusts Bob who trusts Eric, then Alice will also trust Eric. This assumes that Bob actually tells Alice that he trusts Eric, which is called a *recommendation*.

It can be shown that trust is not always transitive in real life [2]. For example the fact that Alice trusts Bob to look after her child, and Bob trusts Eric to fix his car, does not imply that Alice trusts Eric for looking after her child, or for fixing her car. However, under certain semantic constraints [15], trust can be transitive, and a trust system can be used to derive trust. In the last example, trust transitivity collapses because the scopes of Alice's and Bob's trust are different.

We define *trust scope*[1] as the specific type(s) of trust assumed in a given trust relationship. In other words, the trusted party is relied upon to have certain qualities, and the scope is what the trusting party assumes those qualities to be.

Let us assume that Alice needs to have her car serviced, so she asks Bob for his advice about where to find a good car mechanic in town. Bob is thus trusted by Alice to know about a good car mechanic and to tell his honest

[*]Support from the ARC Research Network Secure Australia acknowledged.

[†]The work reported in this paper has been funded in part by the Co-operative Research Centre for Enterprise Distributed Systems Technology (DSTC) through the Australian Federal Government's CRC Programme (Department of Education, Science, and Training).

[1]The terms "trust context" [6], "trust purpose" [13] and "subject matter" [20] have been used in the literature with the same meaning.

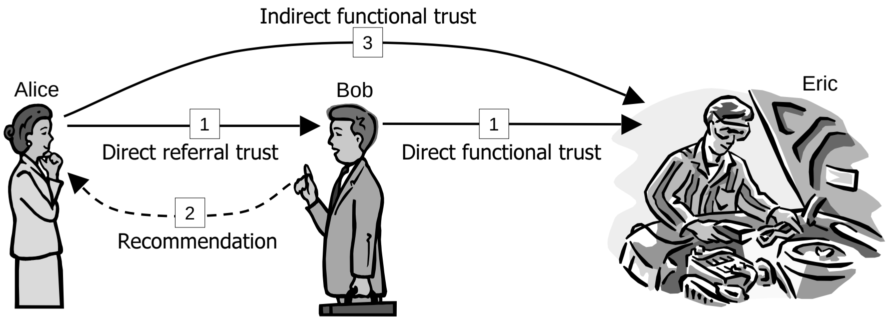


Figure 1: Transitive trust principle

opinion about that. Bob in turn trusts Eric to be a good car mechanic. This situation is illustrated in Fig.1, where the indexes indicate the order in which the trust relationships and recommendations are formed.

It is important to separate between trust in the ability to recommend a good car mechanic which represents *referral trust*, and trust in actually being a good car mechanic which represents *functional trust*. The scope of the trust is nevertheless the same, namely to be a good car mechanic. Assuming that, on several occasions, Bob has proved to Alice that he is knowledgeable in matters relating to car maintenance, Alice's referral trust in Bob for the purpose of recommending a good car mechanic can be considered to be *direct*. Assuming that Eric on several occasions has proved to Bob that he is a good mechanic, Bob's functional trust in Eric can also be considered to be direct. Thanks to Bob's advice, Alice also trusts Eric to actually be a good mechanic. However, this functional trust must be considered to be *indirect*, because Alice has not directly observed or experienced Eric's skills in car mechanics.

Let us slightly extend the example, wherein Bob does not actually know any car mechanics himself, but he knows Claire, whom he believes knows a good car mechanic. As it happens, Claire is happy to recommend the car mechanic named Eric. As a result of transitivity, Alice is able to derive trust in Eric, as illustrated in Fig.2, where the indexes indicate the order in which the trust relationships and recommendations are formed. The prefix "dr-" denotes direct referral trust, "df-" denotes direct functional trust, and "if-" denotes indirect functional trust.

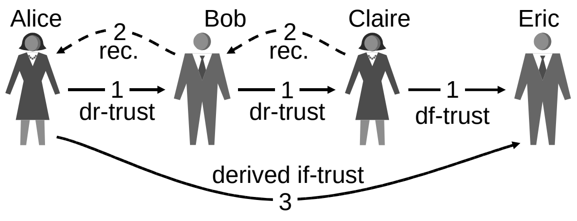


Figure 2: Transitive serial combination of trust arcs

Defining the exact scope of Alice's trust in Bob is more complicated in the extended example. It is most obvious to say that Alice trusts Bob to recommend somebody (who can recommend somebody etc.) who can recommend a good car mechanic. The problem with this type of formulation is that the length of the trust scope expression becomes proportional with the length of the transitive path, so that the trust scope expression rapidly becomes impenetrable. It can be observed that this type of trust scope has a recursive structure that can be exploited to define a more compact expression for the trust scope. As already mentioned, trust in the ability to recommend represents referral trust, and is precisely what allows trust to become transitive. At the same time, referral trust always assumes the existence of a functional trust scope at the end of the transitive path, which in this example is about being a good car mechanic.

The "referral" variant of a trust scope can be considered to be recursive, so that any transitive trust chain, with arbitrary length, can be expressed using only one trust scope with two variants. This principle is captured by the following criterion.

Definition 1 (Functional Trust Derivation Criterion) *Derivation of functional trust through referral trust, requires that the last trust arc represents functional trust, and all previous trust arcs represent referral trust.*

In practical situations, a trust scope can be characterised by being general or specific. For example, knowing how to change wheels on a car is more specific than to be a good car mechanic, where the former scope is a subset of the latter. Whenever a given trust scope is part of all the referral and functional trust scopes in a path, a transitive trust path can be formed based on that trust scope. This can be expressed with the following consistency criterion.

Definition 2 (Trust Scope Consistency Criterion) *A valid transitive trust path requires that there exists a trust scope which is a common subset of all trust scopes in the path. The derived trust scope is then the largest common subset.*

Trivially, every arc in a path can carry the same trust scope. Transitive trust propagation is thus possible with two variants (i.e. functional and referral) of a single trust scope.

Specifying the two scope variants separately can be omitted in case it is difficult to separate between them in a given application. Although trust scopes are always expressed with the two variants in all the descriptions and example of this paper, it is perfectly possible to assume the same descriptions and examples without specifying the two variants.

A transitive trust path stops with the first functional trust arc encountered when there are no remaining outgoing referral trust arcs. It is, of course, possible for a principal to have both functional and referral trust in another principal, but that should be expressed as two separate trust arcs.

The examples above assume some sort of absolute trust between the agents along the transitive trust path. In reality trust is never absolute, and many researchers have proposed to express trust as discrete verbal statements, as probabilities or other continuous measures. One observation which can be made from an intuitive perspective is that trust is diluted through transitivity. Revisiting the example of Fig.2, it can be noted that Alice's trust in the car mechanic Eric through the recommenders Bob and Claire can be at most as confident as Claire's trust in Eric.

It could be argued that negative trust in a transitive chain can have the paradoxical effect of strengthening the derived trust. Take for example the case where Bob distrusts Claire and Claire distrusts Eric, whereas Alice trusts Bob. In this situation, Alice might actually derive positive trust in Eric, since she relies on Bob's advice, and Bob says: *"Claire is a cheater, do not rely on her"*. So the fact that Claire distrusts Eric might count as a pro-Eric argument from Alice's perspective. The question boils down to *"is the enemy of my enemy my friend?"*. However this question relates to how multiple types of untrustworthiness, such as dishonesty and unreliability, should be interpreted in a trust network, which is outside the scope of this study.

3 Parallel Trust Combination

It is common to collect advice from several sources in order to be better informed when making decisions. This can be modelled as *parallel trust combination* illustrated in Fig.3, where again the indexes indicate the order in which the trust relationships and recommendations are formed.

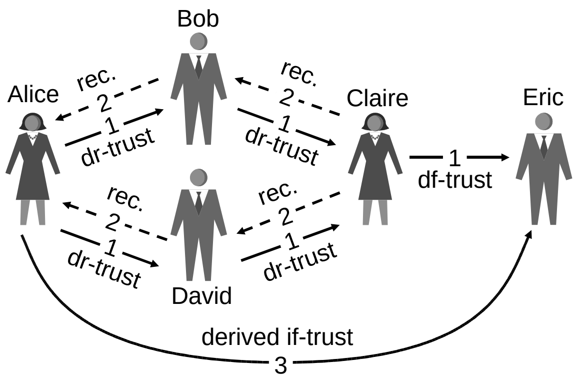


Figure 3: Parallel combination of trust paths

Let us assume again that Alice needs to get her car serviced, and that she asks Bob to recommend a good car mechanic. When Bob replies that Claire, a good friend of his, recommended Eric to him, Alice would like to get a second opinion, so she asks David whether he has heard about Eric. David also knows and trusts Claire, and has heard from her that Eric is a good car mechanic. Alice who does not know Claire personally, is unable to obtain a first hand recommendation about the car mechanic Eric, i.e. she does not directly know anybody with functional trust in Eric. Intuitively, if both Bob and David recommend Claire as a good advisor regarding car mechanics, Alice's trust in Claire's advice will be stronger than if she had only asked Bob. Parallel combination of positive trust thus has the effect of strengthening the derived trust.

In the case where Alice receives conflicting recommended trust, e.g. trust and distrust at the same time, she needs some method for combining these conflicting recommendations in order to derive her trust in Eric. Our method, which is described in Sec.6, is based on subjective logic which easily can handle such cases.

4 Structured Notation

Transitive trust networks can involve many principals, and in the examples below, capital letters A, B, C, D and E will be used to denote principals instead of names such as Alice and Bob.

We will use basic constructs of directed graphs to represent transitive trust networks. We will add some notation elements which allow us to express trust networks in a structured way.

A single trust relationship can be expressed as a directed arc between two nodes that represent the trust source and the trust target of that arc. For example the arc $[A, B]$ means that A trusts B.

The symbol ":" will be used to denote the transitive connection of two consecutive trust arcs to form a transitive trust path. The trust relationships of Fig.2 can be expressed as:

$$([A,E]) = ([A,B] : [B,C] : [C,E]) \quad (1)$$

where the trust scope is implicit. Let the trust scope e.g. be defined as σ: *"trust to be a good car mechanic"*. Let the functional variant be denoted by "fσ" and the referral variant by "rσ". A distinction can be made between initial *direct trust* and derived *indirect trust*. Whenever relevant, the trust scope can be prefixed with "d" to indicate direct trust (dσ), and with "i" to indicate indirect trust (iσ). This can be combined with referral and functional trust, so that for example indirect functional trust can be denoted as "ifσ". A reference to the trust scope can then be explicitly included in the trust arc notation as e.g. denoted by $[A, B, \mathrm{dr}\sigma]$. The trust network of Fig.2 can then be explicitly expressed as:

$$\begin{aligned} &([A,E,\mathrm{if}\sigma]) = \\ &\quad ([A,B,\mathrm{dr}\sigma] : [B,C,\mathrm{dr}\sigma] : [C,E,\mathrm{df}\sigma]) \end{aligned} \quad (2)$$

Let us now turn to the combination of parallel trust paths, as illustrated in Fig.3. We will use the symbol "$\diamond$" to denote the graph connector for this purpose. The "$\diamond$" symbol visually resembles a simple graph of two parallel paths between a pair of agents, so that it is natural to use it for this purpose. Alice's combination of the two parallel trust paths from her to Eric in Fig.3 is then expressed as:

$$\begin{aligned} ([A,E,\mathrm{if}\sigma]) = \ & ((([A,B,\mathrm{dr}\sigma] : [B,C,\mathrm{dr}\sigma]) \diamond \\ & ([A,D,\mathrm{dr}\sigma] : [D,C,\mathrm{dr}\sigma])) \ : \\ & [C,E,\mathrm{df}\sigma]) \end{aligned} \quad (3)$$

In short notation, the same trust graph is expressed as:

$$\begin{aligned} ([A,E]) = \ & ((([A,B] : [B,C]) \diamond \\ & ([A,D] : [D,C])) : [C,E]) \end{aligned} \quad (4)$$

It can be noted that Fig.3 contains two paths. The graph consisting of the two separately expressed paths would be:

$$\begin{aligned} ([A,E]) = \ & ([A,B] : [B,C] : [C,E]) \diamond \\ & ([A,D] : [D,C] : [C,E]) \end{aligned} \quad (5)$$

A problem with Eq.(5) is that the arc $[C, E]$ appears twice. Although Eq.(4) and Eq.(5) consist of the same two paths, their combined structures are different. Some computational models would be indifferent to Eq.(4) and Eq.(5), whereas others would produce different results depending on which expression is being used. When implementing the serial ":" as binary logic "AND", and the parallel "$\diamond$" as binary logic "OR", the results would be equal. However, when implementing ":" and "$\diamond$" as probabilistic multiplication and comultiplication respectively, the results would be different. It would also be different in the case of applying subjective logic operators for transitivity and parallel combination which will be described in Sec.6 below. In general, it is therefore desirable to express graphs in a form where an arc only appears once. This will be called a *canonical expression*.

Definition 3 (Canonical Expression) *An expression of a trust graph in structured notation where every arc only appears once is called canonical.*

With this structured notation, arbitrarily large trust networks can be explicitly expressed in terms of source, target, and scope, as well as other attributes such as measure and time whenever required.

A general directed trust graph is based on directed trust arcs between pairs of nodes. With no restrictions on the possible trust arcs, trust paths from a given source X to a given target Y can contain cycles, which could result in inconsistent calculative results. Cycles in the trust graph must therefore be controlled when applying calculative methods to derive measures of trust between two parties. *Normalisation* and *simplification* are two different control approaches. Our model is based on graph simplification, and a comparison with normalisation methods used in e.g. PageRank proposed by Page *et al.* (1998) [21], and in EigenTrust proposed by Kamvar *et al.* (2003) [17] is provided in [10].

5 Network Simplification

Simplification of a trust network consists of including as many arcs as possible from the original trust network, while still maintaining a canonical expression. Graphs that can be represented as canonical expressions with our structured notation are known as *directed series-parallel graphs* (DSPG) [5]. A DSPG can be constructed by sequences of serial and parallel compositions that are defined as follows [5]:

Definition 4 (Directed Series-Parallel Composition)

- *A* directed series *composition consists of replacing an arc $[A, C]$ with two arcs $[A, B]$ and $[B, C]$ where B is a new node.*
- *A* directed parallel *composition consists of replacing an arc $[A, C]$ with two arcs $[A, C]_1$ and $[A, C]_2$.*

The principle of directed series and parallel composition are illustrated in Fig.4.

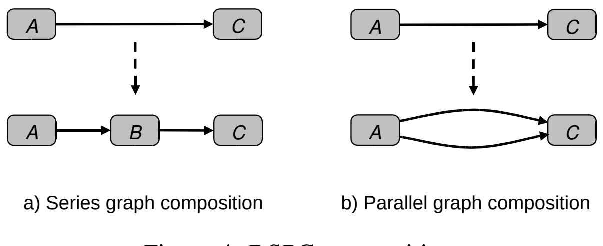


Figure 4: DSPG composition.

By successively applying the principles of series and parallel composition, arbitrarily large DSPGs can be constructed.

We will first describe an algorithm for determining all practical trust paths from a given source to a given target, and secondly algorithms for determining near-optimal or optimal DSPGs.

5.1 Finding Paths

The first step is to determine the possible directed paths between a given pair of agents called the start source and the final target. The pseudo-code in Fig.5 represents an algorithm for finding all practical directed paths between a given start source and a given final target, where no single path contains cycles.

```
Pseudo-Constructor for a trust arc between two parties:

Arc(Node source, Node target, Scope scope, Variant variant){
    this.source = source;
    this.target = target;
    this.scope = scope;
    this.variant = variant;
}

Pseudo-code for a depth-first path finding algorithm:
After completion, 'paths' contains all possible paths between source
and target.

void FindPaths(Node source, Node target, Scope scope) {
    SELECT arcs FROM graph WHERE (
        (arcs.source == source) AND
        (arcs.target NOT IN path) AND
        (arcs.scope == scope))
    FOR EACH arc IN arcs DO {
        IF (
                (arc.target == target) AND
                (arc.variant == 'functional') AND
                (Confidence(path + arc) > Threshold)) {
            paths.add(path + arc);
        }
        ELSE IF (
                (arc.target != target) AND
                (arc.variant == 'referral') AND
                (Confidence(path + arc) > Threshold)) {
            path.add(arc);
            FindPaths(arc.target, target, scope);
            path.remove(arc);
        }
    }
}

Pseudo-code for method call:
The global variables 'path' and 'paths' are initialized.

Vector path = NEW Vector OF TYPE arc;
Vector paths = NEW Vector OF TYPE path;
FindPaths(StartSource, FinalTarget, scope);
```

Figure 5: Path finding algorithm

In the pseudocode of Fig.5, the conditional

$$\text{IF (Confidence(path + arc)} > \text{Threshold)}$$

represents a heuristic rule for simplifying the graph analysis, where the path is only retained as long as the conditional is TRUE. By removing paths with low confidence, the number of paths to consider is reduced while the information loss can be kept to an insignificant level. For a given application, the threshold can be defined as the lowest level for which a trust relationship is meaningful. The mathematical interpretation of confidence is described in Sec.6.1.

5.2 Finding Directed Series-Parallel Graphs

Ideally, all the possible paths discovered by the algorithm of Fig.5 should be taken into account when deriving the trust value. A general directed graph will often contain cycles and dependencies. This can be avoided by excluding certain paths, but this can also cause information loss.

Specific selection criteria are needed in order to find the optimal subset of paths to include.

Fig.6 illustrates an example of a non-DSPG with dependent paths, where it is assumed that A is the source and E is the target. While there can be a large number of possible distinct paths, it is possible to use heuristic rules to discard paths, e.g. when their confidence drops below a certain threshold.

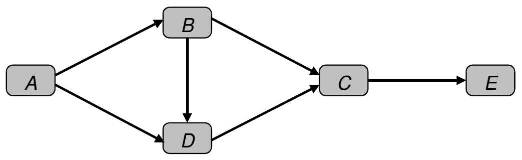


Figure 6: Dependent paths

With n possible paths, there are $2^n - 1$ different combinations for constructing graphs, of which not all necessarily are DSPGs. Of the graphs that are DSPGs, only one will be selected for deriving the trust measure.

In Fig.6 there are 3 possible paths between A and E:

$$\begin{aligned} \phi_1 &= ([A,B] : [B,C] : [C,E]), \\ \phi_2 &= ([A,D] : [D,C] : [C,E]), \\ \phi_3 &= ([A,B] : [B,D] : [D,C] : [C,E]). \end{aligned} \tag{6}$$

This leads to the following 7 potential combinations/graphs.

$$\begin{array}{lll} \gamma_1 = \phi_1, & \gamma_4 = \phi_1 \diamond \phi_2, & \gamma_7 = \phi_1 \diamond \phi_2 \diamond \phi_3. \\ \gamma_2 = \phi_2, & \gamma_5 = \phi_1 \diamond \phi_3, & \\ \gamma_3 = \phi_3, & \gamma_6 = \phi_2 \diamond \phi_3, & \end{array} \tag{7}$$

The graph represented by γ_7 contains all possible paths between A and E. The problem with γ_7 is that it can not be represented as a canonical expression, i.e. where an arc can only appear once. In this example, one path must must be removed from the graph in order to have a canonical expression. The expressions γ_4, γ_5 and γ_6 can be canonicalised, and the expressions γ_1, γ_2 and γ_3 are already canonical, which means that all the expressions except γ_7 can be used as a basis for constructing a DSPG and for deriving A's trust in E.

The optimal DSPG is the one that results in the highest confidence level of the derived trust value. This principle focuses on maximising certainty in the trust value, and not e.g. on deriving the most positive or negative trust value. The interpretation of confidence can of course have different meanings depending on the computational model, and our approach is based on he classic confidence value of probability density functions.

There is a trade-off between the time it takes to find the optimal DSPG, and how close to the optimal DSPG a simplified graph can be. It is possible to use a relatively fast *heuristic algorithm* to find a DSPG close to, or equal to the optimal DSPG. It is also possible to use a relatively slow *exhaustive algorithm* that is guaranteed to find the optimal DSPG.

5.2.1 Heuristic Search for Near-Optimal DSPGs

Fig.7 represents a heuristic algorithm for finding a near-optimal DSPG. It constructs the DSPG by including new paths one by one in decreasing order of confidence. Each new path that potentially could turn the graph into a non-DSPG and break canonicity is excluded. This is detected by analysing each new potential branch with the method:

dspg.sep_subgraph(branch.source,branch.sink)

```
Pseudo-code search algorithm for a near optimal DSPG:
After completion, 'dspg' contains a near-optimal trust graph

void FindNearOptimalDSPG(Vector paths) {
    paths.sort_according_to_confidence;
    dspg = paths(0);
    paths.remove(0);
    FOR EACH path IN paths DO {
        end_of_path = FALSE;
        branch = EMPTY;
        WHILE NOT end_of_path DO {
            next_arc = path.next;
            end_of_path = path.no_more_arcs;
            IF (next_arc.sink NOT IN dspg) {
                branch.add(next_arc);
            }
            ELSE IF ((next_arc.sink IN dspg) AND
                    (branch != EMPTY)) {
                branch.add(next_arc);
                IF (dspg.sep_subgraph(branch.source,branch.sink) {
                    dspg.add(branch);
                    branch = EMPTY;
                }
                ELSE {
                    end_of_path = TRUE;
                }
            }
        }
    }
}

Pseudo-code for method call:
The global variables 'dspg' and 'paths' are initialized.

Vector dspg = NEW Vector OF TYPE arc;
Vector paths = NEW Vector OF TYPE path;
FindNearOptimalDSPG(paths);
```

Figure 7: Heuristic algorithm for a near-optimal DSPG

which returns TRUE if the new branch can be added, and FALSE if not. More precisely, it verifies that the subgraph between the nodes where the new branch is to be added is a separate sub-DSPG, so that a clean parallel graph composition according to Fig.4 is possible when adding the new branch. While this subgraph analysis can be computationally intensive, efficiency can be improved by caching these intermediate results, so that in case several new branches between the same nodes must be added, the analysis of the corresponding subgraph only needs to be done once.

This method only requires the computation of the trust value for a single DSPG, with computational complexity $\mathrm{Comp} = lm$, where m is average number of paths in the DSPGs, and l is the average number of arcs in the paths.

The heuristic method produces a DSPG with overall confidence in the trust level equal or close to that of the optimal DSPG. The reason why this method can not guarantee to produce the optimal DSPG, is that it could exclude two or more paths with relatively low confidence levels because of conflict with a single path with high confidence level previously included, whereas the low confidence paths together could provide higher confidence than the previous high confidence path alone. In such cases it would have been optimal to exclude the single high confidence path, and instead include the low confidence paths. However, only the exhaustive method described below can guarantee to find the optimal DSPG in such cases.

5.2.2 Exhaustive Search for the Optimal DSPG

The exhaustive method of finding the optimal DSPG consists of determining all possible DSPGs, then deriving the trust value for each one of them, and finally selecting the

DSPG and the corresponding canonical expression that produces the trust value with the highest confidence level.

For brevity, we have not included the pseudocode algorithm for the exhaustive search algorithm, because it would be similar to the heuristic search algorithm. The main difference is that all $2^n - 1$ possible orders of including the paths are tried one by one, potentially leading to $2^n - 1$ different DSPGs that must be evaluated. Normally, the DSPG that produces the highest confidence is finally selected.

The computational complexity of the exhaustive method is $\text{Comp} = lm(2^n - 1)$, where n is the number of possible paths, m is the average number of paths in the DSPGs, and l is the average number of arcs in the paths.

6 Trust Derivation with Subjective Logic

Subjective logic represents a practical belief calculus that can be used for calculative analysis trust networks. TNA-SL requires trust relationships to be expressed as beliefs, and trust networks to be expressed as DSPGs in the form of canonical expressions. In this section we describe how trust can be derived with the belief calculus of subjective logic. A numerical example is given in Sec.7.

6.1 Subjective Logic Fundamentals

Belief theory is a framework related to probability theory, but where the probabilities over the set of possible outcomes do not necessarily add up to 1, and the remaining probability is assigned to the union of possible outcomes. Belief calculus is suitable for approximate reasoning in situations of partial ignorance regarding the truth of a given proposition.

Subjective logic [7] represents a specific belief calculus that uses a belief metric called *opinion* to express beliefs. An opinion denoted by $\omega_x^A = (b, d, u, a)$ expresses the relying party A's belief in the truth of statement x. When a statement for example says *"Party X is honest and reliable regarding σ"*, then the opinion about the truth of that statement can be interpreted as trust in X within the scope of σ. Here b, d, and u represent belief, disbelief and uncertainty respectively, where $b, d, u \in [0, 1]$ and $b + d + u = 1$. The confidence parameter used in the pseudocode of Fig.fig:find-path can be defined as equal to $(1 - c)$, i.e. the confidence of a trust value is equivalent to the certainty of the corresponding opinion. The parameter $a \in [0, 1]$ is called the base rate, and is used for computing an opinion's probability expectation value that can be determined as $\mathrm{E}(\omega_x^A) = b + au$. More precisely, a determines how uncertainty shall contribute to the probability expectation value $\mathrm{E}(\omega_x^A)$. In the absence of any specific evidence about a given party, the base rate determines the *a priori* trust that would be put in any member of the community.

The opinion space can be mapped into the interior of an equal-sided triangle, where, for an opinion $\omega_x = (b_x, d_x, u_x, a_x)$, the three parameters b_x, d_x and u_x determine the position of the point in the triangle representing the opinion. Fig.8 illustrates an example where the opinion about a proposition x from a binary state space has the value $\omega_x = (0.7,\ 0.1,\ 0.2,\ 0.5)$.

The top vertex of the triangle represents uncertainty, the bottom left vertex represents disbelief, and the bottom right vertex represents belief. The parameter b_x is the value of a linear function on the triangle which takes value 0 on the edge which joins the uncertainty and disbelief vertexes and takes value 1 at the belief vertex. In other words, b_x is equal to the quotient when the perpendicular distance between the opinion point and the edge joining the uncertainty and disbelief vertexes is divided by

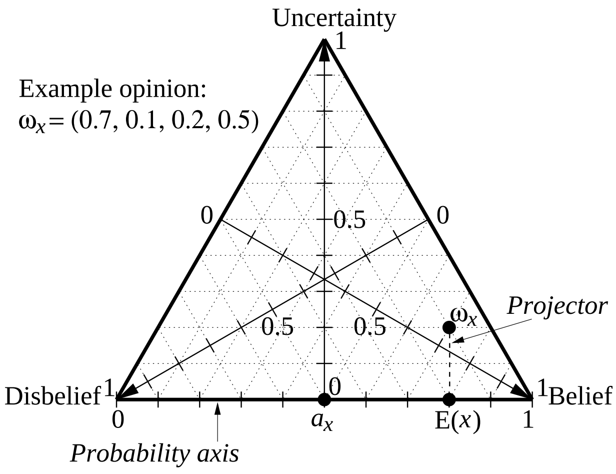


Figure 8: Opinion triangle with example opinion

the perpendicular distance between the belief vertex and the same edge. The parameters d_x and u_x are determined similarly. The base of the triangle is called the probability axis. The base rate is indicated by a point on the probability axis, and the projector starting from the opinion point is parallel to the line that joins the uncertainty vertex and the base rate point on the probability axis. The point at which the projector meets the probability axis determines the expectation value of the opinion, i.e. it coincides with the point corresponding to expectation value $\mathrm{E}(\omega_x^A)$.

Opinions can be ordered according to probability expectation value, but additional criteria are needed in case of equal probability expectation values. We will use the following rules to determine the order of opinions [7]:

Let ω_x and ω_y be two opinions. They can be ordered according to the following rules by priority:

1. The opinion with the greatest probability expectation is the greatest opinion.

2. The opinion with the least uncertainty is the greatest opinion.

3. The opinion with the least base rate is the greatest opinion.

The probability density over binary event spaces can be expressed as beta PDFs (probability density functions) denoted by beta (α, β) [3]. Let r and s express the number of positive and negative past observations respectively, and let a express the *a priori* or base rate, then α and β can be determined as:

$$\alpha = r + 2a\ , \qquad \beta = s + 2(1 - a)\ . \tag{8}$$

The following bijective mapping between the opinion parameters and the beta PDF parameters can be determined analytically [7, 14].

$$\left\{\begin{array}{l} b_x = r/(r+s+2) \\ d_x = s/(r+s+2) \\ u_x = 2/(r+s+2) \\ a_x = \text{base rate of } x \end{array}\right. \iff \left\{\begin{array}{l} r = 2b_x/u_x \\ s = 2d_x/u_x \\ 1 = b_x + d_x + u_x \\ a = \text{base rate of } x \end{array}\right. \tag{9}$$

This means for example that a totally ignorant opinion with $u_x = 1$ and $a_x = 0.5$ is equivalent to the uniform PDF beta $(1, 1)$ illustrated in Fig.9.

It also means that a dogmatic opinion with $u_x = 0$ is equivalent to a spike PDF with infinitesimal width and infinite height expressed by beta $(b_x\eta,\ d_x\eta)$, where $\eta \to \infty$. Dogmatic opinions can thus be interpreted as being based on an infinite amount of evidence.

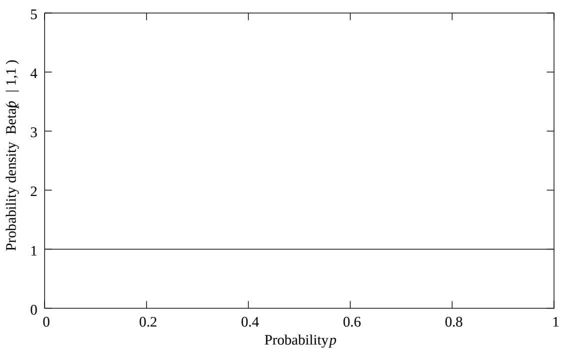


Figure 9: *A priori* uniform beta(1,1)

After r positive and s negative observations in case of a binary state space (i.e. $a = 0.5$), the *a posteriori* distribution is the beta PDF with $\alpha = r + 1$ and $\beta = s + 1$. For example the beta PDF after observing 7 positive and 1 negative outcomes is illustrated in Fig.10, which also is equivalent to the opinion illustrated in Fig.8

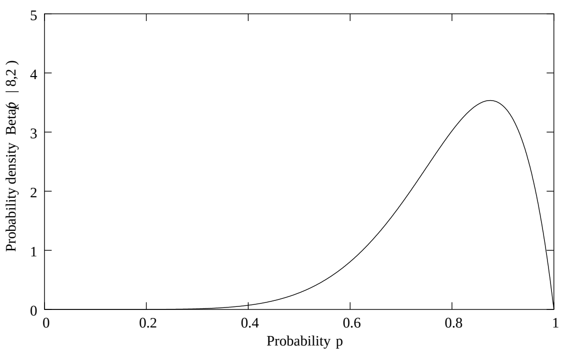


Figure 10: *A posteriori* beta(8,2) after 7 positive and 1 negative observations

A PDF of this type expresses the uncertain probability that a process will produce positive outcome during future observations. The probability expectation value of Fig.10 is $\mathrm{E}(p) = 0.8$. This can be interpreted as saying that the relative frequency of a positive outcome in the future is somewhat uncertain, and that the average value is 0.8.

The variable p is a probability variable, so that for a given p the probability density beta(α, β) represents second order probability. The first-order variable p represents the probability of an event, whereas the density beta(α, β) represents the probability that the first-order variable has a specific value. Since the first-order variable p is continuous, the second-order probability beta(α, β) for any given value of $p \in [0, 1]$ is vanishingly small and therefore meaningless as such. It is only meaningful to compute $\int_{p_1}^{p_2}$ beta(α, β) for a given interval $[p_1, p_2]$, or simply to compute the expectation value of p. The expectation value of the PDF is always equal to the expectation value of the corresponding opinion. This provides a sound mathematical basis for combining opinions using Bayesian updating of beta PDFs.

6.2 Determining Trust with Reputation Systems

The trust representation of subjective logic is directly compatible with the reputation representation of Bayesian reputation systems [12, 13, 25, 24]. This makes it possible to use reputation systems to determine trust measures. The method for doing this is briefly described below.

Bayesian reputation systems allow agents to rate other agents, both positively and negatively, by arbitrary amounts, for a single transaction. This rating takes the form of a vector:

$$\rho = \begin{bmatrix} r \\ s \end{bmatrix}, \text{ where } r \geq 0 \text{ and } s \geq 0. \qquad (10)$$

A simple binary rating system can e.g. be implemented by using $\rho^+ = [1, 0]$ for a satisfactory transaction and $\rho^- = [0, 1]$ for an unsatisfactory transaction [11].

A particular rating can be denoted as:

$$\rho^X_{Z,t_R} \qquad (11)$$

which can be read as X's rating of Z at time t_R. Whenever not relevant, these super- and subscripts can be omitted.

6.2.1 Aging Ratings

Agents (and in particular human agents) may change their behaviour over time, so it is desirable to give greater weight to more recent ratings. This can be achieved by introducing a longevity factor λ, which controls the rate at which old ratings are 'forgotten':

$$\rho^{X,t}_{Z,t_R} = \lambda^{t-t_R} \rho^X_{Z,t_R} \qquad (12)$$

where $0 \leq \lambda \leq 1$, t_R is the time at which the rating was collected and t is the current time.

6.2.2 Aggregating Ratings

Ratings may be aggregated by simple addition of the components (vector addition).

For each pair of agents (X, Z), an aggregate rating $\rho^t(X, Z)$ can be calculated that reflects X's overall opinion of Z at time t:

$$\rho^t(X, Z) = \sum \rho^{X,t}_{Z,t_R}, \text{ where } t_R \leq t\ . \qquad (13)$$

Also, Z's aggregate rating by all agents in a particular set S can be calculated:

$$\rho^t(Z) = \sum_{X \in S} \rho^t(X, Z). \qquad (14)$$

In particular, the aggregate rating for Z, taking into account ratings by the entire agent community C, can be calculated:

$$\rho^t(Z) = \sum_{X \in C} \rho^t(X, Z). \qquad (15)$$

6.2.3 The Reputation Score

Once aggregated ratings for a particular agent are known, it is possible to calculate the reputation probability distribution for that agent. This also takes into account the base rate reputation score a of all agents in the community. The reputation score is then expressed as:

$$\mathrm{beta}(\rho^t(Z)) = \mathrm{beta}(r + 2a,\ s + 2(1 - a)), \qquad (16)$$

where

$$\rho^t(Z) = \begin{bmatrix} r \\ s \end{bmatrix}.$$

However probability distributions, while informative, cannot be easily interpreted by users. A simpler point estimate of an agent's reputation is provided by $\mathrm{E}[\ \mathrm{beta}(\rho^t(Z))\ ]$, the expected value of the distribution. This provides a score in the range $[0, 1]$, which can be scaled to any range (including, for example, '0% reliable to 100% reliable').

Definition 5 (Reputation Score) *Let $\rho^t(Z) = [r, s]'$ represent target Z's aggregate ratings at time t. Then the function $R^t(Z)$ defined by:*

$$R^t(Z) = \mathrm{E}[\,\mathrm{beta}(\rho^t(Z))\,] = \frac{r + 2a}{r + s + 2} \tag{17}$$

is called Z's reputation score at time t.

The reputation score $R^t(Z)$ can be interpreted as a probability measure indicating how a particular agent is expected to behave in future transactions.

The base rate a is particularly useful for determining the reputation score of agents for which the aggregated ratings have low confidence, e.g. because the agents have been idle for longer periods, or because they are new entrants to the community. It is interesting to note that in a community where the base rate a is high, a single negative rating will influence the reputation score more than a single positive rating. Similarly, in a community where the base rate a is low, a single positive rating will influence the reputation score more than a single negative rating. This nicely models the intuitive observation from everyday life where *"it takes many good experiences to balance out one bad experience"*.

6.3 Trust Reasoning

Subjective logic defines a number of operators [7, 22, 16], where some represent generalisations of binary logic and probability calculus operators, whereas others are unique to belief theory because they depend on belief ownership. Here we will only focus on the *discounting* and the *consensus* operators. The discounting operator can be used to derive trust from transitive paths, and the consensus operator can be used to derive trust from parallel paths. These operators are described below.

- **Discounting** [7] is used to compute transitive trust. Assume two agents A and B where A has referral trust in B, denoted by $\omega^A_B = (b^A_B, d^A_B, u^A_B, a^A_B)$. In addition B has functional trust in C, denoted by $\omega^B_C = (b^B_C, d^B_C, u^B_C, a^B_C)$. A's indirect functional trust in C can then be derived by discounting B's trust in C with A's trust in B. The derived trust is denoted by $\omega^{A:B}_C = (b^{A:B}_C, d^{A:B}_C, u^{A:B}_C, a^{A:B}_C)$. By using the symbol '$\otimes$' to designate this operator, we can write $\omega^{A:B}_C = \omega^A_B \otimes \omega^B_C$.

$$\begin{cases} b^{A:B}_C = b^A_B b^B_C \\ d^{A:B}_C = b^A_B d^B_C \\ u^{A:B}_C = d^A_B + u^A_B + b^A_B u^B_C \\ a^{A:B}_C = a^B_C\ . \end{cases} \tag{18}$$

  The effect of discounting in a transitive path is to increase uncertainty, i.e. to reduce the confidence in the expectation value.

- **Consensus** [7, 8, 9] is used to fuse two (possibly conflicting) beliefs into one. Let $\omega^A_C = (b^A_C, d^A_C, u^A_C, a^A_C)$ and $\omega^B_C = (b^B_C, d^B_C, u^B_C, a^B_C)$ be trust in C from A and B respectively. The opinion $\omega^{A\diamond B}_C = (b^{A\diamond B}_C, d^{A\diamond B}_C, u^{A\diamond B}_C, a^{A\diamond B}_C)$ is then called the consensus between ω^A_C and ω^B_C, denoting the trust that an imaginary agent $[A, B]$ would have in C, as if that agent represented both A and B. By using the symbol '$\oplus$' to designate this operator, we can write $\omega^{A\diamond B}_C = \omega^A_C \oplus \omega^B_C$.

  Case I: $u^A_C + u^B_C - u^A_C u^B_C \neq 0$

$$\begin{cases} b^{A\diamond B}_C = \frac{b^A_C u^B_C + b^B_C u^A_C}{u^A_C + u^B_C - u^A_C u^B_C} \\ d^{A\diamond B}_C = \frac{d^A_C u^B_C + d^B_C u^A_C}{u^A_C + u^B_C - u^A_C u^B_C} \\ u^{A\diamond B}_C = \frac{u^A_C u^B_C}{u^A_C + u^B_C - u^A_C u^B_C} \\ a^{A\diamond B}_C = a^A_C \end{cases}$$

  Case II: $u^A_C + u^B_C - u^A_C u^B_C = 0$

$$\begin{cases} b^{A\diamond B}_C = (\gamma^{A/B}\, b^A_C + b^B_C)/(\gamma^{A/B} + 1) \\ d^{A\diamond B}_C = (\gamma^{A/B}\, d^A_C + d^B_C)/(\gamma^{A/B} + 1) \\ u^{A\diamond B}_C = 0 \\ a^{A\diamond B}_C = a_C\ . \end{cases}$$

  where the relative weight $\gamma^{A/B} = \lim(u^B_C/u^A_C)$

  The effect of the consensus operator is to reduce uncertainty, i.e. to increase the confidence in the expectation value. In case the subjective opinions are probability values ($u = 0$), Case II produces the weighted average of probabilities.

The discounting and consensus operators will be used for the purpose of deriving trust measures in the example below.

7 Example Derivation of Trust Measures

Transitive trust graphs can be stored and represented in a computer system in the form of a list of directed trust arcs with additional attributes.

This numerical example is based the trust graph of Fig.3. Table 1 specifies trust measures expressed as opinions. The DSTC Subjective Logic API[2] was used to compute the derived trust values.

Table 1: Direct trust measures of Fig.3

Arc	Variant	Measure	Time
$[A, B]$	r	(0.9, 0.0, 0.1, 0.5)	τ_1
$[A, D]$	r	(0.9, 0.0, 0.1, 0.5)	τ_1
$[B, C]$	r	(0.9, 0.0, 0.1, 0.5)	τ_1
$[C, E]$	f	(0.9, 0.0, 0.1, 0.5)	τ_1
$[D, C]$	r	(0.3, 0.0, 0.7, 0.5)	τ_1
$[A, B]'$	r	(0.0, 0.9, 0.1, 0.5)	τ_2

A parser based on the algorithms of Fig.5 and Fig.7 can go through the arcs of Table 1 to construct the trust

[2]Available at `http://security.dstc.com/spectrum/`

network of Fig.3, and the corresponding canonical expression of Eq.(4). By applying the discounting and consensus operators to the expression of Eq.(4), a derived indirect trust measure can be computed.

- Case a:

  First assume that A derives her trust in E at time τ_1, in which case the first entry for $[A, B]$ is used. The expression for the derived trust measure and the numerical result is given below.

$$\begin{aligned}\omega^A_E &= ((\omega^A_B \otimes \omega^B_C) \oplus (\omega^A_D \otimes \omega^D_C)) \otimes \omega^C_E \\ &= (0.74,\ 0.00,\ 0.26,\ 0.50)\end{aligned} \tag{19}$$

  The derived trust measure can be translated into a beta PDF according to Eq.(9) and visualised as a density function as illustrated by Fig.11

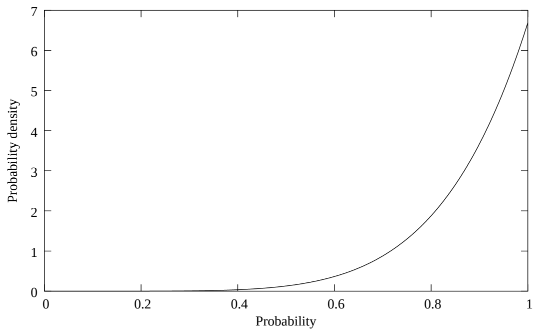


Figure 11: $\omega^A_E \equiv$ beta$(6.7,\ 1.0)$

- Case b:

  Let us now assume that based on new experience at time τ_2, A's trust in B suddenly is reduced to that of the last entry for $[A, B]$ in Table 1. As a result of this, A needs to update her derived trust in E and computes:

$$\begin{aligned}\omega'^A_E &= ((\omega'^A_B \otimes \omega^B_C) \oplus (\omega^A_D \otimes \omega^D_C)) \otimes \omega^C_E \\ &= (0.287,\ 0.000,\ 0.713,\ 0.500)\end{aligned} \tag{20}$$

  Fig.12 below visualises the derived trust measure.

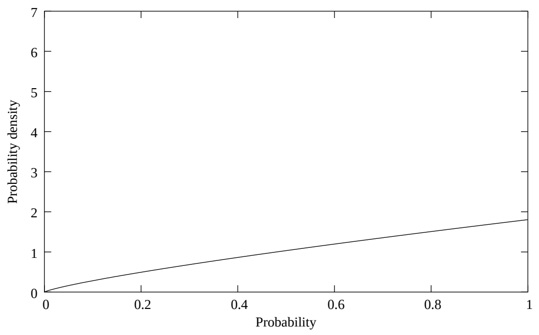


Figure 12: $\omega'^A_E \equiv$ beta$(1.8,\ 1.0)$

It can be seen that the trust illustrated in Fig.11 is relatively strong but that the trust in Fig.12 approaches the uniform distribution of Fig.9 and therefore is very uncertain. The interpretation of this is that the distrust introduced in the arc $[A, B]$ in case (b) has rendered the path $([A, B] : [B, C] : [C, E])$ useless. In other words, when A distrusts B, then whatever B recommends is completely discounted by A. It is as if B had not recommended anything at all. As a result A's derived trust in E must be based on the path $([A, D] : [D, C] : [C, E])$ which was already weak from the start.

8 Discussion and Conclusion

We have presented a notation for expressing trust networks, and a method for trust network analysis based on graph simplification and trust derivation with subjective logic. This approach is called Trust Network Analysis with Subjective Logic (TNA-SL).

Our approach is different from trust network analysis based on normalisation, as e.g. in PageRank and EigenTrust. The main advantage of normalisation is that large highly connected random graphs can be analysed while still taking all arcs into account. The main disadvantages of normalisation is that it is difficult to express negative trust, and that it makes trust measures relative, which prevents them from being interpreted in any absolute sense like e.g. statistical reliability. Neither PageRank nor EigenTrust can handle negative trust.

Trust network simplification with TNA-SL produces networks expressed as directed series-parallel graphs. A trust arc has the three basic attributes of source, target and scope, where the trust scope can take either the functional or the referral variant. This makes it possible to express and analyse fine-grained semantics of trust. Additionally, we have incorporated the attributes of measure and time into the model in order to make it suitable for deriving indirect trust measures through computational methods.

One advantage of TNA-SL is that negative trust can be explicitly expressed and propagated. In order for distrust to be propagated in a transitive fashion, all intermediate referral arcs must express positive trust, with only the last functional arc expressing negative trust. Another advantage is that trust measures in our model are equivalent to beta PDFs, so that trust measures can be directly interpreted in statistical terms, e.g. as measures of reliability. This also makes it possible to consistently derive trust measures from statistical data. Our model is for example directly compatible with Bayesian reputation systems [12, 25], so that reputation scores can be directly imported as trust measures. This rich way of expressing trust separates between the nominal trust value (positive/negative) and the confidence level (high/low), and also carries information about the baseline trust in the community.

The main disadvantage of TNA-SL is that a complex and cyclic network must be simplified before it can be analysed, which can lead to loss of information. While the simplification of large highly connected networks could be slow, heuristic techniques can significantly reduce the computational effort. This is done by ignoring paths for which the confidence level drops below a certain threshold, and by including the paths with the strongest confidence level first when constructing a simplified network. This also leads to minimal loss of information.

The approach to analysing transitive trust networks described here provides a practical method for expressing and deriving trust between peers/entities within a community or network. It can be used in a wide range of applications, such as monitoring the behaviour of peers and assisting decision making in P2P communities, providing a quantitative measure of quality of web services, assessing the reliability of peers in Internet communities, and evaluating the assurance of PKI certificates. Combined with subjective logic, TNA-SL allows trust measures to be efficiently analysed and computed, and ultimately interpreted by humans and software agents.

References

[1] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Conference on Security and Privacy*, Oakland, CA, 1996.

[2] B. Christianson and W. S. Harbison. Why Isn't Trust Transitive? In *Proceedings of the Security Protocols International Workshop*. University of Cambridge, 1996.

[3] M.H. DeGroot and M.J. Schervish. *Probability and Statistics (3rd Edition)*. Addison-Wesley, 2001.

[4] C. Ellison et al. *RFC 2693 - SPKI Certification Theory*. IETF, September 1999. url: http://www.ietf.org/rfc/rfc2693.txt.

[5] P. Flocchini and F.L. Luccio. Routing in Series Parallel Networks. *Theory of Computing Systems*, 36(2):137–157, 2003.

[6] T. Grandison and M. Sloman. A Survey of Trust in Internet Applications. *IEEE Communications Surveys and Tutorials*, 3, 2000.

[7] A. Jøsang. A Logic for Uncertain Probabilities. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 9(3):279–311, June 2001.

[8] A. Jøsang. The Consensus Operator for Combining Beliefs. *Artificial Intelligence Journal*, 142(1–2):157–170, October 2002.

[9] A. Jøsang, M. Daniel, and P. Vannoorenberghe. Strategies for Combining Conflicting Dogmatic Beliefs. In Xuezhi Wang, editor, *Proceedings of the 6th International Conference on Information Fusion*, 2003.

[10] A. Jøsang, E. Gray, and M. Kinateder. Simplification and Analysis of Transitive Trust Networks (to appear). *Web Intelligence and Agent Systems*, 00(00):00–00, 2005.

[11] A. Jøsang, S. Hird, and E. Faccer. Simulating the Effect of Reputation Systems on e-Markets. In P. Nixon and S. Terzis, editors, *Proceedings of the First International Conference on Trust Management (iTrust)*, Crete, May 2003.

[12] A. Jøsang and R. Ismail. The Beta Reputation System. In *Proceedings of the 15th Bled Electronic Commerce Conference*, Bled, Slovenia, June 2002.

[13] A. Jøsang, R. Ismail, and C. Boyd. A Survey of Trust and Reputation Systems for Online Service Provision (to appear). *Decision Support Systems*, 00(00):00–00, 2006.

[14] A. Jøsang and S. Pope. Normalising the Consensus Operator for Belief Fusion. In *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence*, Sydney 2005.

[15] A. Jøsang and S. Pope. Semantic Constraints for Trust Tansitivity. In S. Hartmann and M. Stumptner, editors, *Proceedings of the Asia-Pacific Conference of Conceptual Modelling (APCCM) (Volume 43 of Conferences in Research and Practice in Information Technology)*, Newcastle, Australia, February 2005.

[16] Audun Jøsang, Simon Pope, and Milan Daniel. Conditional deduction under uncertainty. In *Proceedings of the 8th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005)*, 2005.

[17] S.D. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the Twelfth International World Wide Web Conference*, Budapest, May 2003.

[18] Liberty-Alliance. *Liberty ID-FF Architecture Overview.* Version: 1.2-errata-v1.0. http://www.projectliberty.org/specs/liberty-idff-arch-overview-v1.2.pdf, 2003.

[19] Liberty-Alliance. *Liberty Trust Models Guidelines.* http://www.projectliberty.org/specs/liberty-trust-models-guidelines-v1.0.pdf, Draft Version 1.0-15 edition, 2003.

[20] G. Mahoney, W. Myrvold, and G.C. Shoja. Generic Reliability Trust Model. In A. Ghorbani and S. Marsh, editors, *Proceedings of the 3rd Annual Conference on Privacy, Security and Trust*, St.Andrews, New Brunswick, Canada, October 2005.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[22] Simon Pope and Audun Jøsang. Analsysis of competing hypotheses using subjective logic. In *Proceedings of the 10th International Command and Control Research and Technology Symposium (ICCRTS)*. United States Department of Defense Command and Control Research Program (DoDCCRP), 2005.

[23] Ronald L. Rivest and Butler Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO'96 Rumpsession, 1996.

[24] R. Wishart, R. Robinson, J. Indulska, and A. Jøsang. SuperstringRep: Reputation-enhanced Service Discovery. In *Proceedings of the* 28th *Australasian Computer Science Conference (ACSC2005)*, 2005.

[25] A. Withby, A. Jøsang, and J. Indulska. Filtering Out Unfair Ratings in Bayesian Reputation Systems. *The Icfain Journal of Management Research*, 4(2):48–64, 2005.

[26] WS-Trust. *Web Services Trust Language (WS-Trust).* ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf, February 2005.

A Semantic Approach to Boost Passage Retrieval Effectiveness for Question Answering

Bahadorreza Ofoghi
John Yearwood
Ranadhir Ghosh

Centre for Informatics and Applied Optimization
School of Information Technology and Mathematical Sciences
University of Ballarat
PO Box 663, Ballarat, Victoria 3353, Australia

bofoghi@students.ballarat.edu.au
{j.yearwood, r.ghosh}@ballarat.edu.au

Abstract

In the current state of the rapid growth of information resources and the huge number of requests submitted by users to existing information retrieval systems; recently, Question Answering systems have attracted more attention to meet information needs providing users with more precise and focused retrieval units. As one of the most challenging and important processes of such systems is to retrieve the best related text excerpts with regard to the questions, we propose a novel approach to exploit not only the syntax of the natural language of the questions and texts, but also the semantics relayed beneath them via a semantic question rewriting and passage retrieval task. The semantic structure used to address the surface mismatch of the semantically related passages and queries is FrameNet which is a lexical resource for English constituted based on frame semantics. We have run our proposed approach on a subset of the TREC 2004 factoid questions to retrieve passages containing correct answers from the AQUAINT collection and we have obtained promising results.

Keywords: Passage Retrieval, FrameNet, Question Answering, Semantic Boosting.

1 Introduction

In recent years, Question Answering (QA) systems have evolved out of the field of Information Retrieval (IR) to better understand and more precisely cope with information requests. Unlike simple and popular keyword-based information retrieval systems (e.g. Web search engines), QA systems aim to communicate directly with users through a natural language which brings more convenience and comprehension to users who submit their information needs. Having received natural language questions, such systems perform various processes to return actual direct answers to the requests eliminating the burden of query formulation and reading lots of irrelevant documents to reach the desired answer by users. This is due to the fact that a user usually wants not whole documents but brief answers to the specific questions like: *"How old is the President? Who was the second person on the moon? When was the storming of the Bastille?"* (Hovy, Gerber et al. 2001).

In a typical architecture of a question answering system, there are four main procedures; i) question analysis and query formulation, ii) document retrieval, iii) passage retrieval, and iv) answer extraction. The task of analysis of a question contains different sub-procedures based on the general view of the question answering system. In an ontology-based system, this consists of finding related ontology nodes for the submitted question in order to carry out further related processes (Hejazi, Mirian et al. 2003), while in most other systems the procedure of question analysis tries to find named entities and/or to recognize the answer category of the question (Moschitti and Harabagiu 2004), to take into account the temporal issues of the question (Saquete, Martinez-Barco et al. 2004), and to formulate the best representative keyword-based query to boost the retrieval precision in the tasks of document and passage retrieval (Brill, Dumais et al. 2002). Obviously, none of these goals could be achieved before precise and sophisticated natural language processing on the question. In the next step the question answering system is supposed to find the best textual documents from inside the collection which is the answer resource of the system. Such documents should contain passages relevant to the topic of the question. The task of document retrieval, which could be automated using the best known search engines, is bypassed in some question answering systems as they retrieve best passages directly from inside the whole collection. However, the main idea of retrieving the most relevant text snippets to the question is commonly accepted by all question answering systems, When it comes to answering specific information needs of users, the successful extraction of candidate and actual answers could be achieved only on the part of the text which is most similar to the queries formulated based on the original questions. The idea of how to find candidate and actual answers of a question is mostly dependent on the syntactic or semantic structure that is used by the question answering system. START tries to extract such short amounts of information based on ternary expressions matching (Katz 1997). There is a proposed idea for modelling documents based on recognizing Named Entities (Pérez-Coutiño, Solorio et al. 2004) which leads to finding corresponding named entities already recognized inside the text using the SUMO ontology. One of the sophisticated approaches to extract answers has been developed based on frame semantics and sentence annotation using the English lexicon resource, FrameNet, which performs frame and frame element matching and makes inferences inside the related parts of the conceptual graph of FrameNet (Narayanan and Harabagiu 2004).

While working on a question answering architecture, we realized that the precision of best known passage retrieval algorithms could not go higher than a low pick due to some inconsistencies between the questions and the contents of the documents. Having considered that the passage retrieval task is one of the necessary sub-processes in a question answering system (Clarke and Terra 2003), it is worthy to work more on this step to boost the current state-of-the-art of the existing best-known passage retrieval algorithms. Hence, we propose and explore a novel approach on boosting the effectiveness of the passage retrieval task in the context of question answering in a large collection of text so that the system could cope with different types of syntactical mismatch between formulated queries and the texts. We justify our approach based on the results we obtained for a subset of the TREC 2004 factoid questions and the AQUAINT collection using the MultiText (Clarke, Cormack et al. 1997) passage retrieval algorithm and Lemur's passage retrieval engine. Our idea, which exploits Intra-Frame relations between different English terms inside the frames of FrameNet (Baker, Fillmore et al. 1998), has been developed on the basis of poor coverage of the two above-mentioned passage retrieval techniques on the answers of the questions. It has shown impressive results, even though the idea requires that the question (rewritten question) be submitted to the passage retrieval engine more than once.

This paper is organised as follows. Section 2 describes what we mean by passage retrieval for question answering, and also introduces the two passage retrieval algorithms that we have used. In section 3 the main idea of Intra-Frame analysis in FrameNet in order to rewrite the questions and retrieve semantically related passages, as well as the methodology of judging the passages, are described. Section 4 explains the experimental issues and finally, in section 5 we conclude the paper.

2 Passage Retrieval for Question Answering

There are different reasons for a question answering system to perform either well or poorly on the basis of the precision of the answers it provides to submitted questions. We are convinced that in order to find candidate answers that can be used to decide about the actual answer, such systems should be provided with one or more text snippets each of which may contain one or more sentences. This is a crucial sub-process of an end-to-end question answering system. It is also clear that in case there is no candidate or actual answer present inside retrieved passages, then there is no chance for the system to return a correct answer.

There have been many efforts on different passage retrieval algorithms (Tellex, Katz et al. 2003) for dissimilar purposes with diverse points of view on the definition of the word "passage". As mentioned in (Callan 1994) and (Kaszkiel and Zobel 1997) and also referred to in (Kaszkiel, Zobel et al. 1999), the most effective and reliable definition of passage is what includes a fixed-length sequence of words starting and ending anywhere in the document. However, it is not clear that they have tried all well-known algorithms including MultiText algorithm (Clarke, Cormack et al. 1997) which, in our experiments, outperforms Lemur's passage retrieval engine (using its best retrieval model) that will be discussed further later. All of the Lemur's passage retrieval models take into account fixed-size passages to be indexed and retrieved.

The output of the passage retrieval task is very dependent on the query formulation of the original question, and certainly, the query formulation process could not be established before accurate knowledge about the index structure (e.g. if phrase indexing is supported, and if stemmed terms are indexed) of the texts inside the collection. In the next sections, we explore the two passage retrieval methods that we used as well as the specific settings necessary for each. The selection of these two passage retrieval algorithms is strongly based on the fact they cover both fixed-size and dynamic-size passages which is of important characteristics of such algorithms.

2.1 MultiText Algorithm

One of the best-known passage retrieval algorithms is the MultiText algorithm exploited for document ranking and retrieval purposes as well. This algorithm interprets all documents as a series of continuous words and also interprets passages as any number of words starting and ending anywhere inside the documents of a collection (Clarke, Cormack et al. 1997). These passages, which are initially identified by covers, start with one of the query keywords and end with another one, not overlapping the boundaries of documents which constitute the unique string of the words. Experiments performed in (Tellex, Katz et al. 2003) show that this algorithm has shown quite high performance; the third highest MRR (Main Reciprocal Rank) in documents retrieved by the PRISE search engine and the highest MRR in those retrieved by the Lucene search engine. The results are obtained among the eight passage retrieval algorithms investigated by the authors. This high performance, as well as the frequent participation of MultiText in TREC (Clarke, Cormak et al. 2000), were the main reasons for choosing MultiText as one of our passage retrieval algorithms.

2.2 Lemur's Retrieval Engine

Lemur is a toolkit designed to facilitate research in language modelling and information retrieval[2]. It includes a well-designed and supported implementation of different functionalities for text parsing, indexing, retrieval, summarization, and clustering. We have used the indexing and passage retrieval functions of Lemur. Focusing on passage retrieval, Lemur has seven retrieval models each of which could be applied for both document and passage retrieval tasks; i) the tf/idf model, ii) the Okapi bm25 model, iii) KL-divergence language model based method, iv) the CORI model, v) CORI collection selection model, vi) Cosine similarity model, and vii) Indri structured query language. After comparing the retrieval efficiency of these different models, the CORI collection selection model showed the best performance in retrieving the most related passages for the TREC 2004 factoid questions in the AQUAINT collection. The task of passage retrieval is performed based on fixed-size passages inside the documents, while passages have overlaps equal to half of the size of the passages.

3 Exploiting Intra-Frame Term-Level Relations inside FrameNet

As most passage retrieval algorithms are dependent on the occurrences of exact matches of surface features inside the queries and textual documents, even their state-of-the-art precision of retrieval could not go beyond the limitations which are formed by mentioned syntactic structures. In other words, there is little chance for any such passage retrieval algorithm to return a passage which contains the word "spot" in response to a query containing the keyword "discover", for instance. This is because of the fact that there could not exist any type of syntactic similarity between the two words, though they share similar meaning. The problem could be still more complex to solve, in a state of common concepts rather than meanings. For example, in a scenario of a passage where the word "son" is mentioned, there is no syntactic clue to relate any query containing the word "father" to the passage. Such types of mismatch between query keywords and those which may occur inside the texts lead us to resolve the issue by moving towards the semantics underlying the text. Initially, we have found a solution to this sort of query and passage mismatch by using FrameNet data in a Question Rewriting and re-retrieval of passages inside the collection.

3.1 FrameNet Lexicon Resource

FrameNet is a lexicon resource for English (Baker, Fillmore et al. 1998) whose infrastructure is based on Frame Semantics (Lowe, Baker et al. 1997) which is different with Marvin Minsky's frames. FrameNet contains two main entities to completely model and conceptualize the scenarios and the target words which could be realized in the scenarios. Frames, in the highest level of abstraction within FrameNet, encode the base definitions necessary to understand the semantics and the scene of each contained word. In other words, real-world knowledge about the scenarios and their related properties are encoded inside the Frames (Lowe, Baker et al. 1997). To address this, each Frame contains some Frame Elements as representatives of the different semantic and syntactic roles (valences) regarding a target word inside the Frame. The semantic roles are usually common among all of the words that are inherited from a Frame. This ensures a suitable generalization over the English words which either have similar meanings or share the context and/or the scenario in which they could occur in the sentences of the language.

FrameNet is different from WordNet as it contains not only words with similar meanings, but also higher level concepts of similar scenarios of usage in the real-world. On the other hand, these scenarios are related to each other to model an end-to-end scenario containing some smaller sub-scenarios. The different relation types existing between Frames cover this overview of the different events all of which could be realized by FrameNet.

In addition, FrameNet has more than what are formulated by the Predicate-Argument Structure (Surdeanu, Harabagiu et al. 2003), considering the fact that predicates in the Predicate-Argument structure normally are the verbs of the language and the arguments are formed based on dissimilar roles that the predicate could play in the sentences of the language. Target words of FrameNet are nouns, verbs, and even adjectives of the language.

Given the above considerations, FrameNet is well suited for our proposed idea on the resolution of the passage-query syntactical mismatches.

3.2 Passage-Query Mismatch Resolution

The generalization over conceptual scenarios and their related properties is the main characteristic of FrameNet that we have been interested in for resolving the problem of poor passage retrieval performance in the context of question answering due to the syntactic mismatch between the words inside the collection and the keywords of the queries formulated based on original questions. The semantic generalization applied by FrameNet is playing the role of the lost chain for retrieving semantically related passages in response to the queries.

It should be noticed that, in the context of question answering, not all types of semantic query expansion is of interest regarding the fact that a question answering system has to be capable of answering exact questions with actual direct answers. For instance, it is not realistic to change the original query, formed based on the original question, using WordNet semantic relations which has performed well for document retrieval tasks (Voorhees 1994). It causes the retrieval of more indirectly related passages to the question leading to extracting answers which may not be of interest. This argument does not include the systems which try to identify online relations between concepts of different abstraction levels (e.g. (Moldovan, Harabagiu et al. 2002)) that may result in a beneficial semantic matching of the text of the questions

[2] http://www.lemurproject.org/lemur/overview.html

and passages. On the other hand, applying ontology relations between entities or using fuzzy inclusion relations (Akrivas, Wallace et al. 2002) could result in irrelevant passages to come up in the final ranking of retrieved passages. We argue that these methods are not suitable for answering direct factoid questions; however, they have performed well in different contexts.

In what is called generalization over conceptual scenarios and their related properties, the actual procedure of our proposed idea contains a joint generalization-specialization action which evokes a Frame and then considers one of the related terms that is inherited from the Frame. This generalization-specialization method guarantees the query remains at the same semantic abstraction level of the original question.

While these sorts of passages either could not be retrieved or have a very low similarity measure with the query, the way to boost the performance of the retrieval is to substitute the target word of the question with semantically related ones. This is what we call Intra-Frame Term-Level relation, as the substitution is performed based on the target Frame inside FrameNet and the lexical units (terms) covered by the Frame. Figure 1 depicts what happens in a cycle of boosting the passage retrieval effectiveness via question rewriting in the context of question answering.

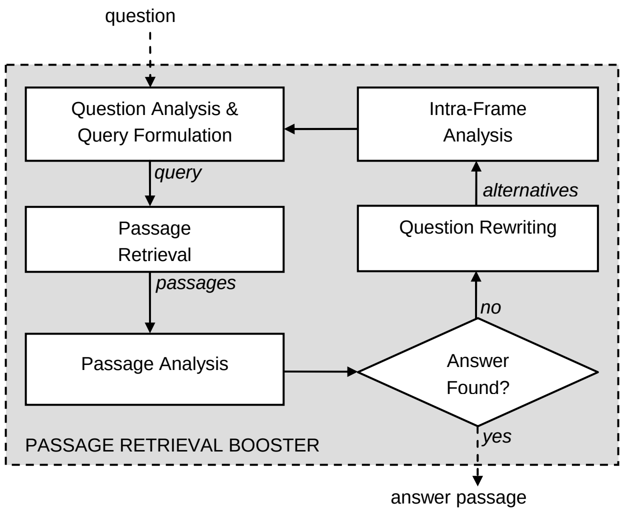


Figure 1: The main cycle of boosting passage retrieval effectiveness in the context of question answering

It should be noticed that the passage retrieval algorithm that is mainly used in this architecture is a modified version of MultiText passage retrieval algorithm whose modifications will be discussed further in the next sections.

The cycle of passage retrieval starts with submitting a question to the system already developed for this purpose. Initially, the question is subject to natural language processing in order that the main keywords to formulate the representative query are known and some other information related to other tasks of question answering is extracted. Then, the query will be sent to the passage retrieval engine to find the best match text excerpts. If the top-ranked passages, based on the manual analysis performed by the passage analysis module, contain the real answer, then no further process is performed at this stage; otherwise, the system tries to identify semantically related text snippets, which are missed due to a syntactic mismatch, after the Intra-Frame analysis on the Frame from which the current target word inherits. The alternative word is one which is also inherited from the evoked Frame by the initial target word and in addition, it has the same part-of-speech (e.g. verb) as that of the initial target. In order to better explain the idea, we consider Example 1.

Example 1: A question from the question list of TREC 2004 is considered (the question id is 3.1 and the target id is 3). The question is fed to the system and the retrieval cycle is as follows;

> Question “When was the comet discovered?” (TREC Target: Hale Bopp comet) → Query “comet discover Hale Bopp” → No Answer in Retrieved Passages → Corresponding Frame Call Evokes the Frame “BECOMING_AWARE” → Intra-Frame Analysis and Alternative Predicate Finding “Spot” → Question Rewriting Using Alternative Predicate “ When was the comet spotted” (TREC Target: Hale Bopp comet) → Query “comet spot Hale Bopp” → Answer Found in the Second Passage.

Inside the AQUAINT collection for TREC 2004, there are some passages containing similar passages to the original question 3.1; however, none of them contains the answer. The top-most passage which is returned by the modified MultiText at the first cycle is:

> <PASSAGE no=1 score= 1.0>
> *Hale-Bopp, a newly-discovered extraordinarily large comet in the solar system, has been recently observed for the first time in China.*
> </PASSAGE>

which is very similar to the query formulated as mentioned above. However, because of the fact that the real answer has not been mentioned using the same predicate “discover”, the passage retrieval algorithm could not either bring the container of the real answer to the top ranks or even retrieve it, as it is the case in this example.

After finding the alternative semantically related predicate “Spot” from inside the corresponding Frame “BECOMING_AWARE”, the rewrite question and the respective query will come up with a passage like below at the second rank;

> <PASSAGE no=2 score= 0.96209>
> *The comet, one of the brightest comets this century, was first spotted by Hale and Bopp, both astronomers in the United States, on July 23, 1995.*
> </PASSAGE>

which contains the correct answer to the question, although it still needs some context resolution and actual answer extraction processes to be performed.

This example clearly shows what happens in the passage retrieval process for question answering systems which could not extract correct answers for those questions which have not a syntactically direct match inside the collection. In contrast, the proposed idea for re-submitting rewrite questions based on Intra-Frame Term-

Level analysis shows promising resolution over the problem.

3.3 Evaluating Passages

As discussed in (Kaszkiel, Zobel et al. 1999) there are usually two ways to measure the retrieval performance of a text retrieval system (e.g. a passage retrieval system). The first way is to measure the *efficiency* which is based on the usage of the resources like disk, time, and memory. In the second manner, the *effectiveness* of the system is measured with regard to the value of satisfaction of users by retrieved texts.

In the context of the question answering systems, the effectiveness of the passages are more important especially to the extent that they potentially deliver correct actual answers to the question submitted by a user.

In focussing on a QA task and using the TREC QA track, our judgment of the passages is based on whether the retrieved passages satisfy the reported correct answer patterns by TREC for each question. In standard passage retrieval, passages are judged for relevance or `aboutness' but in this instance we are assessing passages on whether or not they contain the correct answers. This is a more stringent requirement than relevance. Consequently many highly similar passages, in this context, will not have the actual answer.

The justification of the passages in passage analysis module of the boosting cycle, in further experiments, is to be based on complicated judgements on the candidate answers in the context of a question answering system, although in our first experiments, as mentioned earlier, this has been done manually with regard to the answer patterns reported by TREC. The manual justification of the passages is subject to further study and work with respect to the features of the answer extraction process in an end-to-end question answering system.

In addition, we are concerned about a reasonable method that could extract such answers from inside the potentially correct passages. We do not cover these issues here as they are part of our work in the question answering architecture and the subject of our current and next study.

4 Experimental Issues

We discuss our experimental results with regard to the three aspects of the research study that is being undertaken for semantically answering factoid questions.

4.1 Data

The dataset that has been used for this research study is the TREC 2004 question list and its corresponding text collection of AQUAINT[3]. This collection contains news articles from New York Times News Service (1998-2000), Xinhua News Service (1996-2000), and Associated Press Worldstream News Service (1998-2000).

The question list contains 65 targets and 230 factoid questions (the total number of all type of the questions is 351). We have tried our proposed idea on a subset of this track which contains 20 targets out of 65 and 65 factoid questions out of 230 which is equal to 28.26% of the total number of factoid questions in the TREC 2004 QA track. However, there are 5 questions out of these 65 factoid questions for which no answer could be found in the AQUAINT collection, as a subset of NIL answers reported by TREC (Voorhees 2004). Therefore, we consider a total of 60 questions in our experiments.

4.2 Procedure

In order that a passage retrieval task is performed, in most question answering systems, there is a document retrieval process prior to the passage retrieval task, as mentioned earlier. This should be the case, especially when manipulating a huge-sized collection of text on which a direct passage retrieval task is very complex and time-consuming. Therefore, we used the top-ranked documents reported by TREC for each target[4] to escape the need of the implementation of a document retrieval engine. This ensures that we are convinced of the necessity of a document retrieval stage, although we have not implemented it and benefited from the results from the PRISE information retrieval system via the TREC reports.

We ran two passage retrieval algorithms on the dataset; modified MultiText, which we implemented, and Lemur's passage retrieval engine, where we used the APIs.

In modified MultiText, we create a feature vector for both the passage and the query. Afterwards, we use the Cosine similarity function to measure the similarity value between passages and the query. To find the feature values of the feature vector for the passages we use Equation 1 and to measure the similarity value between the two feature vectors of the query and the passage Equation 2 is applied, which is composed of the well-known Cosine Similarity Function and the effect of the term coverage of the passage.

$$vec_i = \frac{tf_i}{\log(passageLength_j) + tf_i} * weight_i \qquad (1)$$

$$Sim(q, p) = \cos(q, p) * \frac{\operatorname{cov}erage_j}{queryLength} \qquad (2)$$

In Equation 1, tf_i is the raw term frequency of the query term i inside the passage, $weight_i$ is the weight of this term which is assigned based on two considerations; i) the part-of-speech of the term (i.e. the verbs have higher weights than nouns, adjectives, and so on), and ii) the terms which occur inside the TREC target of the question gain a bonus on their weights to increase up to 1.0. The value $coverage_j$, in Equation 2, contains the unique

[3] http://www.ldc.upenn.edu/Catalog/docs/LDC2002T31/

[4] http://trec.nist.gov/data/qa.html

number of the query terms that a passage covers and *queryLength* is the total number of the terms inside the query.

While running Lemur's passage retrieval algorithm, we used the passage size of 160 words. Authors in (Kaszkiel, Zobel et al. 1999) have mentioned that this could be in the optimum value range for the passage retrieval algorithms which take into account a fixed size for the passages to be retrieved. Also, we tried different retrieval methods of Lemur and decided that the CORI-Collection Selection method outperforms the other supported models in the context of our work.

4.3 Results

We developed a software platform to test the two above-mentioned passage retrieval algorithms and also to perceive the increase on the output results based on the evaluation methodology explained at the section 3.3.

As shown in Table 1, the highest retrieval effectiveness for Lemur's retrieval engine, which has been acquired by the CORI-Collection Selection retrieval model, was 58.2%, while this percentage went up to 70% for the same questions using the modified MultiText algorithm.

Retrieval Method	**Questions with Answers in Top 10 Passages**	**No. of Questions**
Lemur's PR	%58.3	60
Modified MultiText	%70	60
Modified MultiText along with Semantic Resolution	%75	60

Table 1: Retrieval effectiveness of the three runs of passage retrieval

The results have been obtained by considering the top 10 passages for each retrieval task. Whenever the answer was recognized inside one of the top 10 passages retrieved for any question the score for that question was considered 1; otherwise 0. In the end, the percentage was calculated as the average value of over all scores.

Because of the higher performance of the MultiText algorithm on the dataset that we are working on, we chose to apply the proposed idea of semantic question rewriting and semantic mismatch resolution on the modified version of the MultiText algorithm. We obtained an effectiveness of 75% on the same subset of factoid questions and their representative queries. A promising increase in effectiveness is gained on a subset of the TREC questions. We expect that this performance may go even higher either on a bigger subset or on the total number of the questions in the track.

5 Conclusion

Due to the poor coverage of the best-known passage retrieval algorithms on the actual answers related to a question answering task of TREC 2004, we have developed an idea to retrieve passages which are not syntactically matched to the keywords of representative queries of the original questions. As long as deep semantic relations are not considered by the passage retrieval process, it can not cope with syntactically mismatched passages which at the same time contain semantically related elements to the question. The proposed idea tries to rewrite the questions which come up in such situations using alternative related terms from inside the evoked Frame of FrameNet by the original target predicate. This rewriting and re-submit cycle is protective of the original semantic abstraction level of the questions and does not cause any unnecessary generalization over the concepts which exist in the questions to avoid retrieving irrelevant passages. We have developed our idea on a subset of the TREC 2004 questions and the AQUAINT collection and have achieved impressive improvement on the state-of-the-art of two best-known passage retrieval algorithms.

6 References

Akrivas, G., M. Wallace, et al. (2002): Context-Sensitive *Semantic* Qquery Expansion. *Proceedings of the 2002 IEEE International Conference on Artificial Intelligence Systems (ICAIS'02).*

Baker, C. F., C. J. *Fillmore*, et al. (1998): The Berkeley FrameNet Project. *International Conference On Computational Linguistics* **1**: 86 - 90.

Brill, E., S. Dumais, et *al*. (2002): An Analysis of the AskMSR Question-Answering System. *Proceedings of 2002 conference on empirical methods in natural language processing (EMNLP 2002).*

Callan, J. P. (*1994*): Passage-Level Evidence in Document Retrieval. *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*: 302-310.

Clarke, C., G. *Cormack*, et al. (1997): Relevance Ranking for One to Three Term Queries. *Proceedings of RIAO-97, 5th International Conference ``Recherche d'Information Assistee par Ordinateur''*: 388-400.

Clarke, C. L. A., G. V. Cormak, et al. (2000): Question Answering By Passage Selection (MultiText Experiments for TREC-9). *Ninth Text REtrieval Conference (TREC 9).*

Clarke, C. L. A. and E. L. Terra (2003): Passage Retrieval vs. Document Retrieval for Factoid Question Answering. *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*: 427 - 428.

Hejazi, M. R., M. S. Mirian, et al. (2003): TeLQAS: A Telecommunication Literature Question Answering System Benefits from a Text Categorization Mechanism. *IKE03*.

Hovy, E., L. Gerber, et al. (2001): Question Answering in Webclopedia. Proceedings *of the TREC-9 Conference*.

Kaszkiel, M. and J. Zobel (1997): Passage Retrieval Revisited. *Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*: 178-185.

Kaszkiel, M., J. Zobel, et al. (1999): Efficient passage ranking for document databases. *ACM Transactions on Information Systems (TOIS)* **17**(4): 406-439.

Katz, B. (1997): Annotating the World Wide Web using Natural Language. *Proceedings of the 5th RIAO Conference on Computer Assisted Information Searching on the Internet (RIAO '97).*

Lowe, J. B., C. F. Baker, et al. (1997): A Frame-Semantic Approach to Semantic Annotation. *SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How?*

Moldovan, D., S. Harabagiu, et al. (2002): LCC Tools for Question Answering. *Proceedings of the eleventh Text REtrieval Conference (TREC 2002).*

Moschitti, A. and S. Harabagiu (2004): A Novel Approach to Focus Identification in Question/Answering Systems. *In proceedings of the Workshop on Pragmatics of Question Answering at HLT-NAACL 2004*.

Narayanan, S. and S. Harabagiu (2004): Question Answering Based on Semantic Structures. *International Conference on Computational Linguistics (COLING 2004).*

Pérez-Coutiño, M., T. Solorio, et al. (2004): Toward a Document Model for Question Answering Systems. *Second International Atlantic Web Intelligence Conference, AWIC 2004*: 145-154.

Saquete, E., P. Martinez-Barco, et al. (2004): Splitting Complex Temporal Questions for Question Answering systems. *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics: ACL 2004*: 566-573.

Surdeanu, M., S. Harabagiu, et al. (2003): Using predicate-argument structures for information extraction. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics* **1**: 8-15.

Tellex, S., B. Katz, et al. (2003): Quantitative Evaluation of Passage Retrieval Algorithms for Question Answering. Proceedings *of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*: 41 - 47.

Voorhees, E. M. (1994): Query Expansion Using Lexical-Semantic Relations. *Annual ACM Conference on Research and Development in Information Retrieval*: 61-69.

Voorhees, E. M. (2004): Overview of the TREC 2004 Question Answering Track. *The Thirteenth Text REtrieval Conference Proceedings (TREC 2004).*

Fillmore, Charles J. (1976): *Frame* semantics and the nature of language. *In Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech* **280**: 20-32.

A Programming Paradigm for Machine Learning, with a Case Study of Bayesian Networks

Lloyd Allison

Clayton School of Information Technology,
Monash University,
Victoria,
Australia 3800.
tel:(+61) 3 9905 5205,
Email: Lloyd.Allison@infotech.monash.edu.au
http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

Abstract

Inductive programming is a new machine learning paradigm which combines functional programming for writing statistical models and information theory to prevent overfitting. Type-classes specify general properties that models must have. Many statistical models, estimators and operators have polymorphic types. Useful operators combine models, and estimators, to form new ones; Functional programming's compositional style of programming is a great advantage in this domain. Complementing this, information theory provides a compositional measure of the complexity of a model from its parts.

Inductive programming is illustrated by a case study of Bayesian networks. Networks are built from classification- (decision-) trees. Trees are built from partitioning functions and models on data-spaces. Trees, and hence networks, are general as a natural consequence of the method. Discrete and continuous variables, and missing values are handled by the networks. Finally the Bayesian networks are applied to a challenging data set on lost persons.
Keywords: inductive inference, functional programming, Haskell, minimum length encoding, statistical models, Bayesian networks.

1 Introduction

The paper describes *inductive programming* (IP) a new paradigm for quickly writing succinct solutions to inductive inference problems from machine learning. Solutions take the form of *statistical models* and their estimators: Given particular data, infer a *general* model from the data; the data are invariably noisy. IP uses functional programming to program models and estimators, and the information theoretic criterion, minimum message length (MML), to prevent over-fitting.

Much research in machine learning involves devising a new kind of statistical model and implementing a program to learn (infer, fit, estimate) a model given data. The resulting stand-alone programs are often hard to modify and to combine with others to implement new statistical models. To address this, IP defines types and classes of statistical models and the properties that instances, that is particular models, must have and provides a library of such instances.

Given the huge variety of problems in general-purpose computing, the chances of having a ready-made program that already solves some new problem is small. Things are no different in inductive inference so it is useful to have a way of creating new solutions, quickly and easily. Programming languages exist to make it easier to write new solutions in general computing. One could devise a special purpose programming language for inductive inference and examples exist, sometimes as a "scripting" language as distinct from the main "implementation" language in a data analysis platform such as R (CRAN 2004) and S-Plus (Crawley 2002). But such scripting languages are often interpreted and lack compile-time type checking. Instead IP uses an existing general purpose functional programming language that is compiled and has a strong type system – Haskell (Peyton-Jones et al 1999). Haskell is a good choice (Allison 2005) for the domain because it is expressive and has a powerful system of polymorphic types and type-classes; it is good programming language technology. Functional programming encourages the *composition* of functions, and polymorphic types lead to general solutions; this makes for short and general programs. We see these benefits rubbing-off on statistical models when they are transformed and composed.

Previous work on IP (Allison 2003) created basic but useful statistical models, estimators and functions. The present paper shows how they can be extended, composed and tailored quickly to suit a new problem, and used as parts of a new model. Many models and associated functions are polymorphic; a good type and class system reveals their true generality. Statistical models and functions on them can be very general – any computable model inferred from almost any type of data by an arbitrary algorithm.

Over-fitting is a well known problem in machine learning. William of Occam argued long ago that an explanation should be kept simple unless necessity dictates otherwise. A computer program doing inductive inference must address model complexity in some way. In particular, if sub-models are to be composed to make new models, the complexity of the parts and the whole must be dealt with. Later we will see basic models used within models of missing data which are used within classification trees which in turn are used within Bayesian networks. With its compositional nature, minimum message length (MML) (section 2.1) inference (Wallace & Boulton 1968, Wallace 2005) is a natural partner for functional programming in machine learning.

The questions that are raised, and that are being answered as IP develops, include: what are the types and classes of statistical models, what can be done to them, and how can they be transformed and

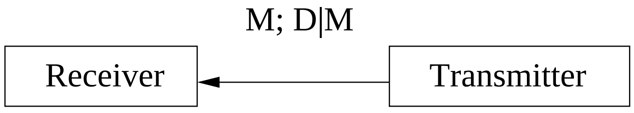


Figure 1: Message Paradigm

combined? Depending on one's background and inclination, IP can be seen as a software engineering analysis of machine learning, as a compositional denotational semantics of statistical models, as an application of functional programming, or as an embedded language (van Deursen, Lint & Visser 2000). The Haskell code produced could also be seen a rapid prototype for other data analysis platforms.

The next section covers background material. After that inductive programming (IP) is illustrated by a case study of Bayesian networks. The Bayesian networks are then applied to a data set of lost persons (Koester 2001). It is a challenging data set of 363 records and 15 variables, half of them missing on average. It shows the kind of problem that typically pops up with real data, if any data set can be said to be typical.

Bayesian networks form a case study; the main aim of the paper is to show how a new statistical model can be programmed quickly to suit a new problem. It explores IP's expressiveness not the statistical performance of any particular model(s). If IP and some other system have equivalent models then those models will, in principle, behave roughly equivalently. Rather the point is to show how IP can be used to create a *new* model to suit a new inference problem, as opposed to "massaging" the problem to suit some existing model.

All code shown is Haskell-98 (Peyton-Jones et al 1999) in the interests of standardization and has been compiled under the Glasgow Haskell Compiler, ghc, version 6.0.1.

2 Background

For completeness, this section briefly introduces MML and IP's main type-classes.

2.1 MML

Minimum message length (MML) (Wallace & Boulton 1968, Wallace 2005) builds on Shannon's mathematical theory of communication (1948), hence 'message', and on Bayes's theorem (Bayes 1763):

```
Pr(M&D) = Pr(M).Pr(D|M) = Pr(D).Pr(M|D)

msgLen(E) = -log(Pr(E))

msgLen(M&D) = msgLen(M)+msgLen(D|M)
            = msgLen(D)+msgLen(M|D)
```

where M is a model (theory, hypothesis, parameter estimate) of prior probability Pr(M) over some data, D, and E is an event of probability Pr(E). MsgLen(E) is the length of a message, in an optimal code, announcing E; the units are *nits* for natural logs, *bits* for base 2 logs.

MML notionally considers a *transmitter* sending a two-part message to a *receiver* (figure 1). The first part, of length msgLen(M), states a model which is an answer to some inference problem. The second part, msgLen(D|M), states the data encoded as if the answer, M, is true; note that the receiver cannot decode the second part without the first part. There is a trade-off between the complexity of the model, M, and its fit to the data, D|M: A simple model is cheap to state but may not fit the data well. A complex model may fit the data better but is more expensive to state (Georgeff & Wallace 1984). In some simple cases MML is equivalent to maximum aposteriori (MAP) estimation but this is not true in general (Wallace & Freeman 1987, Farr & Wallace 2002). For example, if one or more continuous parameters are involved they must be stated to *finite*, optimal precision, and MML shows how to do this. Note that a one-part message can be very slightly more efficient in transmitting D but it does not offer an *explanation* of D; it does not state an answer to an inference question.

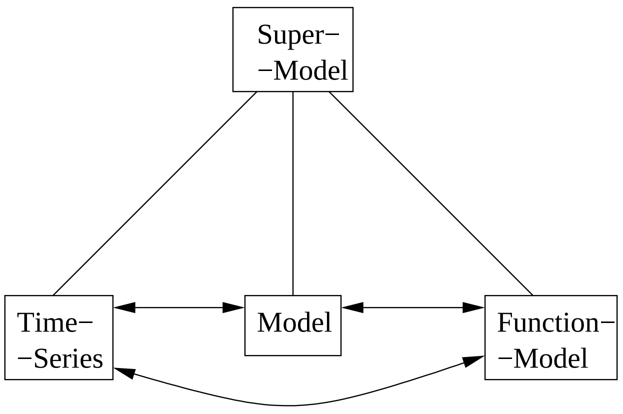


Figure 2: Classes and Conversions

MML (Wallace & Boulton 1968) is related to the minimum description length (MDL) principle (Rissanen 1978). The former aims to select a parameterized model – the parameters being stated to optimal precision – and embraces explicit priors. The latter aims to select a model-class and favours universal distributions and implicit priors. MML and MDL have been featured in the Journal of the Royal Statistical Society (Wallace & Freeman 1987, Rissanen 1987) and in a special issue of the Computer Journal on Kolmogorov complexity (Gammerman & Vovk 1999). Oliver and Baxter (1994, p. 24) made a direct comparison and concluded that only MML (Wallace & Freeman 1987) had all the desirable properties of invariance under non-linear transformations of parameters, of applicability to large and small samples (not only asymptotic), and of making a definite inference.

Strict MML (SMML) relies on the design of a full optimal code book. Unfortunately SMML is infeasible for most inference problems (Farr & Wallace 2002). Fortunately there are efficient, accurate MML approximations for many useful problems and models (Wallace 2005).

MML is a natural *compositional* criterion because the complexity of data, models and sub-models are all measured in the same units. "[It is possible] to use [message] length to select among competing subtheories at some low level of abstraction, which in turn can form the basis (i.e., the 'data') for theories at a higher level of abstraction. There is no guarantee that such an approach will lead to the best global theory, but it is reasonable to expect in most natural domains that the resulting global theory will at least be near-optimal" (Wallace & Georgeff 1983). MML's compositional nature is a good fit with functional programming's compositional style of programming. This is illustrated in the Bayesian network case study of section 3. MML has been used to assess the complexity of combined models of some specific types (e.g. Allison et al (1999) and Powell et al (2004))

```
class ... SuperModel sMdl where
 prior   :: sMdl -> Probability
 msg1    :: sMdl -> MessageLength
 mixture :: ... mx sMdl -> sMdl
 ...

class Model mdl where
 pr   :: (mdl dataSpace) ->
         dataSpace -> Probability
 nlPr :: (mdl dataSpace) ->
         dataSpace -> MessageLength
 msg  :: ... (mdl dataSpace) ->
             [dataSpace] -> MessageLength
 msg2 :: (mdl dataSpace) ->
         [dataSpace] -> MessageLength
 ...

class FunctionModel fm where
 condModel :: (fm inSpace opSpace) ->
              inSpace -> ModelType opSpace
 ...
```

'`...`' stands for omitted details, '`::`' for 'has type', '`[t]`' for 'list of a type `t`', and '`->`' for function type.

Figure 3: Classes of Statistical Model

but its full *programming* potential has only recently started to be studied (Allison 2003). A functional programming language with a parametric polymorphic type system is a sound foundation for such a study.

2.2 Types and Classes of Statistical Models

We want to be able to program as large as possible a set of things that people call *statistical models* and yet have the set clean, orthogonal and built on a small foundation. For simplicity, we also want a small collection of just their essential properties. Here statistical model is taken to cover things that assign explicit probabilities to data. Haskell type-classes (figures 2, 3) `Model`, `FunctionModel` and `TimeSeries` were previously defined (Allison 2003, Allison 2005) for basic models (distributions), function-models (regressions) and time-series models; the first two are used in the following case study.

A basic `Model`, `mdl` (figure 3), can return the probability, `pr`, and the negative log probability, `nlPr`, of a datum from its data-space. It can also compute the second-part, `msg2`, and the total two-part message length, `msg` (section 2.1), for a data set. We are only concerned with the most important properties here; a statistical model might be able to do several other things.

Note that in Haskell `t->u` denotes the function type with input type `t` and output type `u`. A data set over a data-space `ds` has the type `[ds]`, that is 'list of `ds`.' For example, the `pr` operator of class `Model` (figure 3) has the type `(mdl dataSpace) -> dataSpace -> Probability`. That is, given a model over the `dataSpace` and a datum from the `dataSpace` return the probability of the datum.

A function-model has an input-space (exogenous variables) and an output space (endogenous variables). Its principal ability is to return a model of its output space conditional, `condModel`, on a value from the input space.

A super-class, `SuperModel`, states that an instance of one of the various sub-classes must return its own prior probability and message length, `msg1`, and that it must be able to form mixtures; it must also be in the standard class `Show` so that we can print the answers to inference problems.

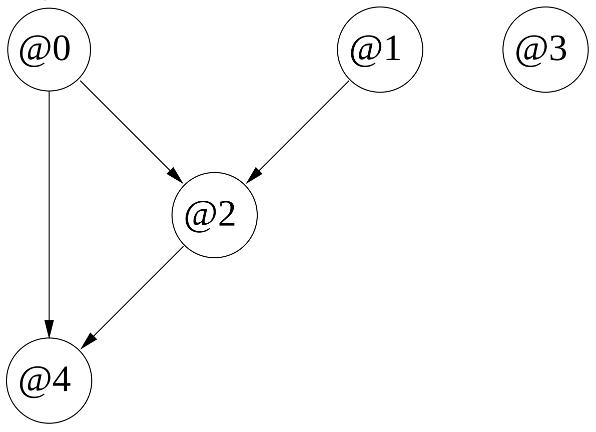


Figure 4: Example Network.

Some types are provided for models to be built in standard ways: Type `ModelType` is an instance of type-class `Model`, and types `FunctionModelType` and `CTreeType` (classification tree type) are instances of type-class `FunctionModel`.

Operators are defined to implement familiar laws of probability. For example, assuming that variables over the data-spaces `ds1` and `ds2` are independent, `bivariate m1 m2` forms a model of the product data-space, `(ds1, ds2)`, from `m1`, a model of `ds1`, and `m2`, a model of `ds2`. For the case where `ds2` is conditionally dependent on `ds1`, `condition m1 fm` forms a model of `(ds1, ds2)` from `m1`, a model of `ds1`, and `fm`, a function-model from `ds1` to `ds2`. There are related operators on estimators – `estBivariate`, `estCondition` and so on. Many of these operators are polymorphic in that their types contain type variables such as `ds1` and `ds2`.

Useful statistical models, including multi-state, integer, normal and multi-variate distributions, mixture models, Markov models, finite function-models (conditional probability tables) and classification trees, have been defined and made instances of the appropriate classes. Below, these building blocks are extended, tested and used in a case study of Bayesian networks to explore and illustrate IP.

3 Case-Study: Bayesian Networks

A Bayesian network (Korb & Nicholson 2004) is a good tool to investigate relationships among the variables of a data set. A Bayesian network is a directed acyclic graph. A node represents a variable. An edge represents a direct conditional dependence of a *child* on a *parent* and, in a suitable context, has a causal interpretation. Creating and applying an estimator of the structure and parameters of a Bayesian network forms our case study to illustrate IP – a network is a non-trivial tool and implementing one provides a good test of a system's expressive power. A Bayesian network is in class `Model` (section 2.2) and can assign a probability to a data tuple; belief updating was not required by the application and has not been implemented. Figure 4 shows an example Bayesian network in which variable 2 is a child of variables 0 and 1 and is a parent of 4, variable 3 has no relationship to the other variables, and so on. It happens that variables 0 and 4 are continuous and variables 1, 2 and 3 are discrete.

Friedman and Goldszmidt (1996) first suggested using decision-trees (classification trees), in place of the full conditional probability tables (CPTs) often used within the nodes of networks over discrete variables; Comley and Dowe (2003) have also used trees within the nodes of networks. A classification tree can "become" a full CPT in the limit but can be much more economical, that is less complex, in many cases.

It happens that previous work created a rather general classification tree algorithm (Allison 2003, Allison 2005). The tree's type is an instance of class `FunctionModel`. Such a tree can test arbitrary variables – discrete, continuous, multi-variate, sequence – from its input space and can have arbitrary distributions over its ouptut space, or even function-models (regressions), in the leaves. These possibilities follow naturally from IP's exploitation of Haskell's polymorphic type system. The classification tree is reused here as the basis of our new Bayesian networks; also see section 3.7. Each classification tree consists of at least one leaf-node, `CTleaf`, and possibly also fork-nodes, `CTfork`. These tree-nodes are *not to be confused* with the network's nodes; there is one tree per network node. The classification tree type is an instance of the class `FunctionModel` (section 2.2). A fork tests a parent (input) variable value. A leaf models the appropriate child (output) variable. Typically the multi-state distribution, `mState`, models a discrete variable, and the normal distribution models a continuous variable but other distributions can be used if desired because the tree estimator is parameterized by the leaf estimator. MML gives a trade-off between the complexity of a tree and its fit to the data and this is used to control the search. Note that when used within a node in a Bayesian network, one or more tests on a parent variable in a tree indicate a parent-child dependency, an edge, at the network level.

The following sections describe the application of Bayesian networks to lost person data. As an example of IP it shows the composition of statistical models: Multi-state and normal distributions within models of missing data within classification trees within a Bayesian network. Some new generic features were required to handle this data set. Any of those features may exist in some other data analysis platform, perhaps this is true of all of them, but it is unlikely that they all exist in the same platform, and unlikely that such a platform could be as easily adapted to further new features. The point is to investigate how easy it is to adapt IP to a new task. This is important because it often seems that every data set has its own oddities as one gets to know it.

3.1 Application of Bayesian Networks: Lost Person

Koester's (2001) lost person data set has been examined in CSSE, Monash (Twardy 2002, Twardy & Hope 2004). Here it provides an application of the Bayesian network case study. There are 363 records, and 15 variables, numbered 0-14. Approximately half of the variable values are missing overall. Attention is sometimes restricted to the first eight variables; one aim is to predict distance travelled, `DistIPP` variable 7, from variables zero to six. In general, workers want to have an "explanation" of the data; the structure and parameters of a network are a good start.

3.2 Describing the Data

The first step in the application is to define the variable types in the lost person data set; in Haskell this is done quite naturally as:

```
data Tipe = Alzheimers| Child| Despondent|
            Hiker| Other| Retarded|
            Psychotic deriving ...
type Age  = Double
data Race = ...
data Gender = ...
data Topography = Mountains| Piedmont|
             Tidewater
             deriving (Ord, Enum, Bounded,...)
data Urban = Rural | Suburban | Urban
             deriving (Ord, Enum, Bounded,...)
type HrsNt   = Double   -- hours notified
type DistIPP = Double   -- distance
...

type MissingPerson =
                (Maybe Tipe, Maybe Age, ... )
```

The Haskell keyword '`deriving`' directs the compiler to add a new data type, for example `Topography`, to standard Haskell classes such as `Ord` (ordered), `Enum` (enumerated) and `Bounded`.

Missing values are an issue and are represented by `Maybe t` where `Maybe t = Nothing | Just t` is a standard Haskell type with parameter t; also see section 3.6.

A datum, a lost person, is a tuple of the component variables. Haskell's standard Prelude (Peyton-Jones et al 1999) instantiates tuples, up to 7-tuples, in classes `Read` and `Show`, so the 15-tuples here need to be made instances of those classes for input and output respectively. This is an easy, if tedious, job and could in principle be automated in template Haskell (Sheard & Peyton-Jones 2002), say.

3.3 Modelling the Variables

The question of which distribution, and therefore which estimator, to use for each variable now arises. The standard estimator for the normal (Gaussian) distribution uses a uniform prior on the mean and an inverse prior on the standard deviation and requires their ranges, and also the data measurement accuracy. Note that the multi-state distribution and its estimator are polymorphic, being applicable to any bounded enumerated data-space (type).

```
e0 = estModelMaybe estMultiState    -- Tipe
e1 = estModelMaybe(estNormal 0 90 1 70 0.5)
...
```

Function `estModelMaybe` was quickly created to allow for missing values in a variable; it is discussed in section 3.6.

Finally the individual estimators are assembled into `estMissingPerson`, a composite that matches a data tuple.

3.4 Partitioning Data Spaces: Class Splits

A classification tree, as used in a node of a Bayesian network, operates by *splitting*, that is partitioning, a data set from its input space by tests on input variables; a `Splitter` partitions a data set. In this way the data are directed into subtrees and eventually into leaves where the output variable(s) can be well modelled. Function `splits` of class `Splits` (Allison 2003) proposes, in order of decreasing prior probability, `Splitters` for use by the classification tree estimator, `estCTree`.

```
class Splits ds where
  splits :: [ds] -> [Splitter ds]
```

The current tree estimator uses a simple zero-lookahead algorithm in the search to balance tree complexity (`msg1`) against fit to the data (`msg2`).

A multi-variate input space is, by default, split by splitting on one of its component variables. To implement this, the ways of splitting the components are interleaved for consideration in turn.

A continuous ordered (Ord) variable, such as Age, is split on being < or ≥ some value. By default splitsOrd proposes values as follows: Median, quartiles, octiles, and so on (Wallace & Patrick 1993).

A discrete, Bounded, enumerated (Enum), variable, such as Gender, of a k-valued type is conventionally split into k parts, as defined in the obvious way by function splitsBE. However Topography and Urban are instances of the standard Haskell classes Bounded, Enum (enumerated) and also Ord (ordered), as can be seen from their definitions, so we also have the options of splitting each of them into two parts on the basis of order, as covered by splitsOrd:

```
instance Splits Tipe where
  splits = splitsBE
...

instance Splits Topography where
  splits = splitsBE
  -- or alternatively  = splitsOrd

instance Splits Urban where
  splits = splitsBE
  -- or alternatively  = splitsOrd
```

Yet another alternative was also implemented and tested for lost persons: Tipe has seven values and high-arity, un-ordered, discrete types like Tipe can cause difficulties to function-models because of the large number of cases and the few data in some or all of them. If some cases are thought to behave in similar ways then, rather than using splitsBE, values can be grouped into nominated sub-sets (and their complement) accordingly. This only affects splitting on Tipe, not modelling of it. A function to implement this setSplits option is just a few lines.

```
setSplits sets [] = []
setSplits sets xs =
 let y:ys = map (memberships sets) xs
 in if all ((==) y) ys then []  --trivial
    else [setSplitter sets]

instance Splits Tipe where  -- e.g. ...
 splits = setSplits [[Alzheimers],[Child]]
```

If ‘k’ sub-sets are specified, their complement is taken to be the (k+1)st. Note that the programmer decides how to group the values in Tipe here. In principle a program could search through the possibilities but it would, of course, add to the overall search time.

It is a simple matter in IP to implement extensions, such as setSplits, to models to suit a problem and its data.

3.5 Selecting Sub-Spaces: Projections

In a typical application of a classification tree, the input variables and the output variable are fixed. But here, in a Bayesian network, the selection of parent (input) and child (output) variables must be under program control on a node by node basis; this property made the case study particularly interesting from the IP point of view. Class Project, as in *projection*, was created for this purpose. Some such mechanism is needed for heterogeneous variable types in a strongly typed language; the network estimator (section 3.7) does not “care” what types the data and sub-estimators have, provided only that they are consistent. An instance, type t, of class Project is some multi-dimensional type for which a list of Boolean flags can be used to restrict t’s activities to certain selected dimensions. The non-selected dimensions behave in a trivial, “identity” manner, that is appropriate to type t, if they are ever called upon. In the case of a Model this behaviour is to return zero information, probability one, for non-selected variables, i.e. they are taken to be already known, or to be of no interest.

```
class Project t where
  select :: [Bool] -> t -> t
  ...
```

As discussed in section 3.4, class Splits exists for partitioning data-spaces – discrete, continuous, multi-variate or whatever other data-spaces are made instances of it. A new class Splits2, inspired by Project, was defined (it could perhaps be folded into class Splits) to allow splitting on only selected variables:

```
class Splits2 ds where
  splitSelect :: [Bool]->[ds]->[Splitter ds]
```

```
estNetwork perm estMV dataSet =
 let
  n = (length . selAll) (estMV [])
  search _  []     = []  -- done
  search ps (c:cs) =
   --parents ps, children c:cs
   let
    opFlag  = ints2flags [c] n  --child
    ipFlags = ints2flags ps  n  --parents
    cTree = estCTree
             (estAndSelect estMV opFlag)
             (splitSelect ipFlags)
             dataSet dataSet
   in cTree : (search (c:ps) cs)

  trees  = search [] perm       --network
  msgLen = sum (map msg1 trees) --total msg1
  nlP datum = sum
   (map(\t -> condNlPr t datum datum) trees)
 in
   MnlPr msgLen nlP        --return a Model
         (\() -> "Net:"++(show trees))
```

Figure 5: Network estimator.

3.6 Handling Missing Data

The lost person data set is difficult in having many missing values. Most data have at least one missing value, and some have several. Every variable is missing in some datum. Haskell already has the ideal type to represent possibly missing values: Maybe. New operators were implemented to extend arbitrary statistical models to cover possibly missing values. Rather than cleaning the data – deleting data with missing values – or imputing (replacing) missing values, missing-ness is built into our model.

Function modelMaybe m1 m2 might be called a “high-order” function on models because it acts on models which are, if not literally functions, principally made up of them. It turns an *arbitrary* model, m2, of non-missing data of type t into the corresponding model of Maybe t. It requires a model, m1, of Bool, for whether the data is present (True) or missing (False).

```
modelMaybe m1 m2 =
 let
  negLogPr (Just x) = nlPr m1 True + nlPr m2 x
  negLogPr Nothing  = nlPr m1 False
 in MnlPr (msg1 m1 + msg1 m2) negLogPr
         ...show method omitted
```

MnlPr is a constructor for a type in class Model; it takes a message length, a negative log likelihood function, and a description which shows the model.

The related high-order function, estModelMaybe acts on estimators; it turns an estimator of non-missing data into the corresponding estimator where the data may include missing values:

```
estModelMaybe estModel dataSet =
 let
  present (Just _) = True
  present Nothing  = False
  m1 = uniformModelOfBool
  m2 = estModel (map (\(Just x)->x)
                (filter present dataSet))
 in modelMaybe m1 m2
```

This is not the same as just coding missing-ness as a "special" value because it is not estimated with the given definition of m1.

In the present application the missing-ness of values is certainly non-random for some variables, for example Age is often not recorded by search teams for cases of Hiker::Tipe. However we are not interested in modelling missing-ness in this problem; it is common knowledge. Hence a fixed unbiased model, m1, is used above to "predict" missing (Nothing) or present (Just...). The following definition of m1 can be used instead if it is necessary to estimate missing-ness:

```
  m1 = estMultiState (map present dataSet)
```

In addition to modelling, missing values also affect splits, that is partitions of the data (section 3.4). A simple strategy is for the variable to be split as for the underlying type but with an extra option for missing (Nothing) cases. Other splitting strategies, not examined here, could try to predict in various ways what the missing value, or its distribution, really is and act on that. There are a great many possibilities and, this being an example, we just give one reasonable, simple approach that is sufficient for the application.

3.7 Mixed Bayesian Networks and the Lost Person Network

The function, estNetwork (figure 5), for inferring a Bayesian network is given a permutation, a total ordering, of the selected variables that are to be considered; a variable may be dependent on none, some or all of the variables preceding it in the permutation. The use of total or partial orders on variables is not uncommon in network learners (Korb & Nicholson 2004). It is sufficient for this application because a plausible ordering of the variables is fairly obvious but, in principle, it would be possible to search over permutations. Such a search would have to be heuristic if there were many variables, and information theory does suggest some heuristics, but the simple algorithm does not do this and the permutation is currently taken to be common knowledge.

Internally estNetwork uses the estimator for classification trees (Allison 2003), estCTree, to do much of the work. The remainder consists of organising selector flags (section 3.5) corresponding to the allowed parents for the child in the current node. Note that the dataSet seems to be passed to estCTree twice, as both the input and output variables – its third and fourth parameters. But its first and second parameters use straightforward auxiliary functions ints2flags, estAndSelect and splitSelect, to flag the child (output) to be predicted by the leaf estimator and the parents (input) to be used for splitting as appropriate at each node in the network.

For lost persons, variables 1 to 3, Age, Race and Gender cannot, in a causal sense, depend on other variables and should come first, in some arbitrary order, say [1,2,3]. Tipe probably depends on one

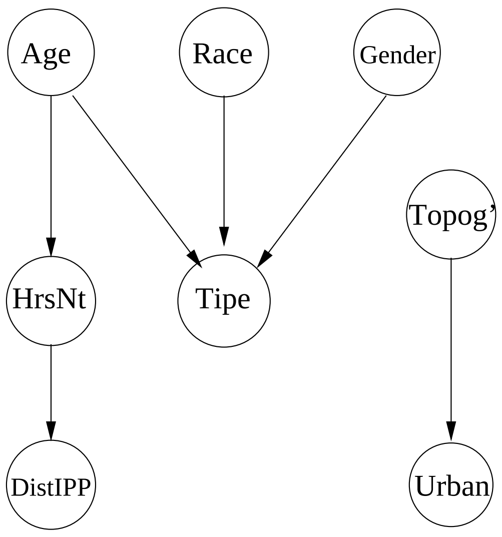


Figure 6: Lost Person Network 1.

or more of them, for example there are few young Alzheimers cases. Topography and Urban can sensibly come next, and one expects a relationship between them. That leaves HrsNt and finally DistIPP to make up a plausible ordering, [1, 2, 3, 0, 4, 5, 6, 7], of the first eight variables. There is also a natural null hypothesis which models the variables independently. The code to run the inference is shown below:

```
dataSet = read (readFile theDataFile)
          :: [MissingPerson]        --input

nw = estNetwork [1,2,3,0,4,5,6,7]
         estMissingPerson dataSet   --model

nullModel = estMissingPerson dataSet
```

Figure 6 shows the first network inferred for variables 0 to 7; the node parameters are inferred but not shown.

Tipe depends strongly on Age and also on Gender and Race. As expected, Urban is dependent on Topography. There is some direct dependence of DistIPP on HrsNt, and of the latter on Age, but there seems to be no strong predictor of DistIPP from other variables. The model is significant with a total two-part message length, for the first eight variables, of 5512 nits against 5936 nits under the null model. Other analyses were tried, for example allowing ordered (Ord) splits on Topography and Urban, in place of Bounded Enum splits; the conclusions were broadly similar.

When Tipe was allowed to split according to setSplits [[Alzheimers], [Child]] (section 3.4), the implicit complement being [Despondent...], the link from Age to HrsNt was replaced by a link from Tipe (which itself depends strongly on Age) for a saving of 6 nits on the model against a loss of 3.7 nits on the data (figure 7). However this small net gain should be taken with a big pinch of salt and may well be due to the pattern of missing data as much as anything.

As a final example, modelling all 15 variables gave the network shown in figure 8; variable Tipe has been duplicated for ease of drawing. The extra variables, 8 to 14, are:
TrackOffset (continuous),
Health = Well | Hurt | Dead,
Outcome = Find | Suspended | Invalid,
FindRes = Ground | Air | Local | Law | Dog,

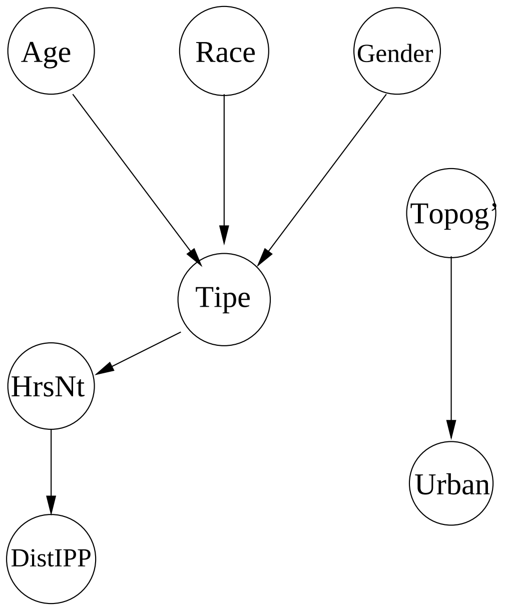


Figure 7: Lost Person Network 2.

`FindLoc = Brush | Woods | Field | Water | Linear | Thing`,
`HrsFind` (continuous),
`HrsTo` (continuous).
These extra variables can all be missing.

3.8 Comparisons

Weka (Witten & Frank 1996) which is based on Java is perhaps the system closest to the present work. Weka's Bayesian networks "assume that all variables are discrete" (Bouckaert 2004) p. 22 and "a limitation of the current classes is that they assume that there are no missing values" (Bouckaert 2004) p. 23. In Weka, continuous variables must be discretized first and how this is done may affect the final result. Discretization is unnecessary in IP for modelling and, for splitting, is part of the network optimization as a by-product of using our classification trees (section 3.2). Missing-ness was built into the model when necessary (section 3.6).

There are distant similarities between IP and inductive logic programming (ILP): There has been some interest in the use of complexity-based measures in ILP (Conklin & Witten 1994, Srinivasan, Muggleton & Bain 1994) but this aspect of ILP is less developed than work on MML. The programmer is involved in the design of the search algorithm (section 3.7) in IP to a greater extent than in ILP, typically in designing new models and estimators; it is infeasible to have a very general search over too large a class of computable statistical models.

A model in IP, particularly one that is used as a component of other models (figure 9), must be able to handle extreme data sets. For example a Bayesian network may contain several trees and each tree may contain several leaf distributions. One or more of those leaf distributions may be given a sub-set of data that is "unusual" – perhaps consisting of just a single item. MML insists that every model effects a valid, decodable message (in principle) so there can be no understating of a model's complexity. A (sub-) model must guarantee this, or at the very least raise an exception if it cannot. This principle keeps us "honest" and ensures that the top-level model's complexity is valid.

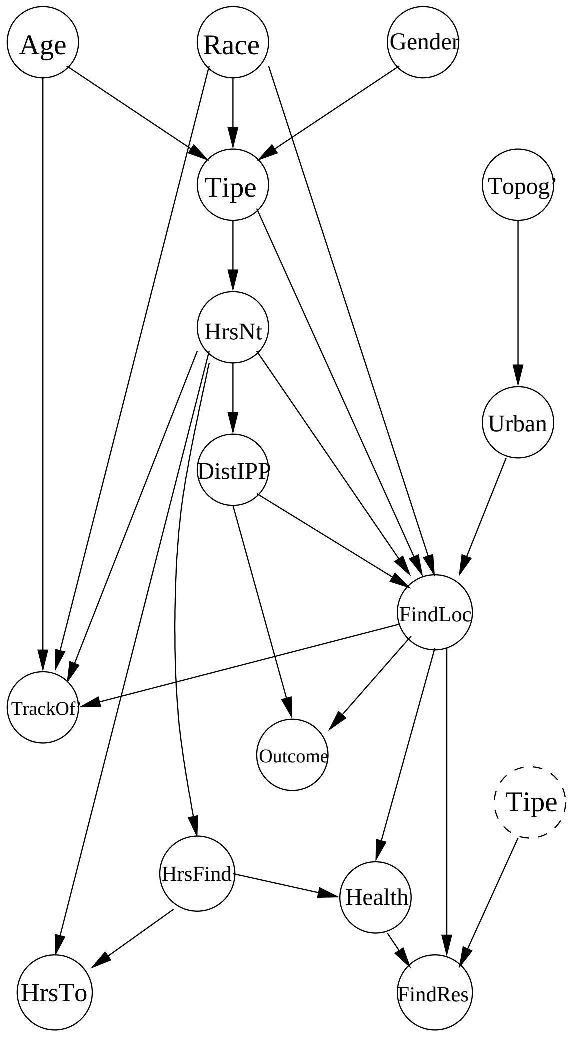


Figure 8: All 15 Variables.

4 Conclusions

Inductive programming (IP) uses the compositional abilities of functional programming, Haskell and minimum message length (MML) inference. Haskell's features have a number of advantages in inductive inference. Mapping a data set, such as lost persons, onto the Haskell type system is a useful exercise in getting to know the data very precisely; a data-analyst will work in this space for some time. The need to define a variable's properties, e.g. `Ord` or not, automatically suggests what is possible, such as whether to split `Topography` as discrete or as ordered data (section 3.2). These things cannot be forgotten; the type and class system brings them to your attention.

The IP code shown is standard Haskell-98 but other experiments (Allison 2004) do show that some Haskell type extensions can be useful in some other problems. In-built support for wide tuples, `(,)`, would make it easier to deal with large multi-variate data sets, although template Haskell (Sheard & Peyton-Jones 2002) is a possible solution.

High-order functions, such as `estModelMaybe` (section 3.6), are invaluable in creating new ways of using *arbitrary* statistical models. The polymorphic type system ensures that the uses are both general and type-safe. Haskell's type inference algorithm often finds a more general type for a function than its programmer did and this can also be the case with statistical models and their estimators. There is potential for an extensive library of operators on statistical

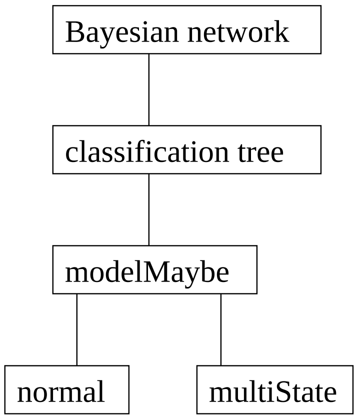


Figure 9: Model Layers

models and their estimators.

Lazy evaluation means, for example, that only models of selected variables of lost persons (section 3.2) are evaluated. Selections are made once at the top level; most of the algorithms do not "consider" the matter at all.

Computing model complexity by minimum message length (section 2.1) is a good match with the compositional style of functional programming. The reader may hardly have noticed any explicit Message length calculations but they are handled by `modelMaybe` (section 3.6) and other functions, and are combined in the complexity of the Bayesian network (section 3.1 and figure 9) and its classification trees (figure 8) to inform the search.

A specific model can be created quickly for a new problem thanks to Haskell's expressive power. Of course it cannot yet be claimed that the types and classes created are the best possible designs for a compositional denotational semantics of statistical models. For example, a case can be made for specifying the notion of a *data set*; perhaps data traversal, data measurement accuracy and data weights should be wrapped up in suitable types and classes. Only more experience and time will let us settle on the best trade-off between generality, usability and efficiency, but experience to date is positive.

The Bayesian network estimator, `estNetwork`, and associated classes `Project` and `Splits2` (section 3.5) took one day to create. The lost person application (section 3.2) came along some weeks later and it took one and a half days to create a working model, including how to handle missing data (section 3.6) which had previously been in the 'must think about that one day' category. Any amount of further time can be spent playing with the data once a model and a program exist, although there is a fine line between data exploration and fishing.

4.1 Acknowledgments.

It is a pleasure to thank Charles Twardy, Ann Nicholson and Kevin Korb for generous discussions of Bayesian networks. Charles Twardy also discussed missing people and he coined the term 'inductive programming'. Chris Wallace (1933–2004) always gave much help and was an inspiration; he is sadly missed. The Department of Computer Science at the University of Wales Aberystwyth, and the Department of Computer Science and the Programming Language and Systems Research Group at the University of York were very hospitable during my visits in 2004.

References

Allison, L. (2003), Types and classes of machine learning and data mining, *in* 26th Australasian Computer Science Conference (ACSC), pp. 207–215.

Allison, L. (2004), Inductive inference 1.1, TR 2004/153, School of Computer Science and Software Engineering, Monash University. http://www.csse.monash.edu.au/~lloyd/tildeFP/II/

Allison, L. (2005), 'Models for machine learning and data mining in functional programming', *J. Functional Programming* **15**(1), pp. 15–32, doi:10.1017/S0956796804005301.

Allison, L., Powell, D. & Dix, T. I. (1999), 'Compression and approximate matching', *BCS Computer J.* **42**(1), pp. 1–10.

Baxter, R. A. & Oliver, J. J. (1994), MDL and MML: Similarities and differences, TR 207, Department of Computer Science, Monash University. (Amended 1995.)

Bayes, T. (1763), 'An essay towards solving a problem in the doctrine of chances', *Phil. Trans. of the Royal Soc. of London* **53**, pp. 370–418, and reprinted in *Biometrika* **45**(3/4), pp. 296–315, 1958.

Bouckaert, R. R. (2004), Bayesian networks in Weka, TR 14/2004, Comp. Sci. Dept.. U. of Waikato.

Comley, J. & Dowe, D. (2003), General Bayesian networks and asymmetric languages, *in* 2nd Hawaii Int. Conf. on Statistics and Related Fields (HICS-2).

Conklin, D. & Witten, I. H. (1994), Complexity-based induction, *Machine Learning* **16**(3), pp. 203–225.

Crawley, M. J. (2002), *Statistical Computing – an Introduction to Data Analysis using S-Plus*, Wiley.

Farr, G. E. & Wallace, C. S. (2002), 'The complexity of strict minimum message length inference', *BCS Computer J.* **45**(3), pp. 285–292.

Friedman, N. & Goldszmidt, M. (1996), Learning Bayesian networks with local structure, *in* Uncertainty in A.I., pp. 252–262.

Gammerman, A. & Vovk, V. (eds) (1999), Special Issue on Kolmogorov Complexity, *BCS Computer J.* **42**(4).

Georgeff, M. P. & Wallace, C. S. (1984), A general selection criterion for inductive inference, *in* European Conf. on Artificial Intelligence (ECAI84), Pisa, pp. 473–482.

Koester, R. J. (2001), 'Virginia dataset on lost-person behaviour', author's site http://www.dbs-sar.com/.

Korb, K. B. & Nicholson, A. E. (2004), *Bayesian Artificial Intelligence*, Chapman and Hall / CRC.

Peyton-Jones, S. et al, (1999), *Report on the Programming Language Haskell-98*, Available at http://www.haskell.org/.

Powell, D. R., Allison, L. & Dix, T. I. (2004), Modelling alignment for non-random sequences, *in* 17th ACS Australian Joint Conf. on Artificial Intelligence (AI2004), Springer-Verlag, LNCS/LNAI Vol. 3339, pp. 203–214.

R, various authors, (2004), 'CRAN: The Comprehensive R Archive Network', http://lib.stat.cmu.edu/R/CRAN/.

Rissanen, J. (1978), 'Modeling by shortest data description', *Automatica* **14**, pp. 465–471.

Rissanen, J. (1987), 'Stochastic complexity', *J. Royal Statistical Society series B.* **49**(3), pp. 223–239 and 252–265.

Shannon, C. E. (1948), 'A mathematical theory of communication', Bell Syst. Technical Jrnl. **27** pp. 379–423 and pp. 623–656.

Sheard, T. & Peyton-Jones, S. (2002), Template meta-programming for Haskell, *in* Proc. of the Workshop on Haskell, ACM, pp. 1–16.

Srinivasan, A., Muggleton, S. & Bain, M. (1994), 'The justification of logical theories based on data compression', *Machine Intelligence* **13**, pp. 87–121.

Twardy, C. R. (2002), 'SAR*bayes*: Predicting lost person behavior', presented to National Association of Search and Rescue (NASAR), available at http://sarbayes.org/nasar.pdf.

Twardy, C. R. & Hope, L. (2004), 'Missing data on missing persons', School of Computer Science and Software Engineering, Monash University, personal communication.

van Deursen, A., Lint, P. & Visser, J. (2000), Domain-specific languages: An annotated bibliography, *in* ACM SIGPLAN Notices **35**(6), pp. 26–36.

Wallace, C. S. (2005), *Statistical and Inductive Inference by Minimum Message Length*, Springer-Verlag.

Wallace, C. S. & Boulton, D. M. (1968), 'An information measure for classification', *BCS Computer J.* **11**(2), pp. 185–194.

Wallace, C. S. & Freeman, P. R. (1987), 'Estimation and inference by compact coding', *J. Royal Statistical Society series B.* **49**(3), pp. 240–265.

Wallace, C. S. & Georgeff, M. P. (1983), A general objective for inductive inference, TR 32, Dept. of Computer Science, Monash University.

Wallace, C. S. & Patrick, J. D. (1993), 'Coding decision trees', *Machine Learning* **11**, pp. 7–22.

Witten, I. H. & Frank E. (1999), Nuts and bolts: Machine learning algorithms in Java, *in* Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, pp. 265–320.

Rule Sets Based Bilevel Decision Model

Zheng Z.[*#], Zhang G.[*], He Q. [#], Lu J.[*], Shi Z.[#]
[*] Faculty of Information Technology, University of Technology, Sydney
PO BOX 123, Broadway, NSW 2007, Australia
# Key Laboratory of Intelligent Information Processing, Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China
`Corresponding Author Email: jielu@it.uts.edu.au`

Abstract

Bilevel decision addresses the problem in which two levels of decision makers, each tries to optimize their individual objectives under constraints, act and react in an uncooperative, sequential manner. Such a bilevel optimization structure appears naturally in many aspects of planning, management and policy making. However, bilevel decision making may involve many uncertain factors in a real world problem. Therefore it is hard to determine the objective functions and constraints of the leader and the follower when build a bilevel decision model. To deal with this issue, this study explores the use of rule sets to format a bilevel decision problem by establishing a rule sets based model. After develop a method to construct a rule sets based bilevel model of a real-world problem, an example to illustrate the construction process is presented.

Keywords: Bilevel programming, Decision making, Decision model, Rough set, Rule set.

1 Introduction

Organizational decision making often involves two levels. In general, the decision maker at the upper level will influence, control or induce the behavior of the decision maker at the lower level but not completely control his action. In addition, the lower level decision maker gains his objective under a given region, although his decision is in a subordinate position. In such a bilevel decision situation, decision maker at each level has individual payoff function, and the upper level the decision maker is at, the more important and global his decision is. Therefore, a bilevel decision model intends to reach certain goals, which reflect the upper level decision makers' aims and also consider the reaction of the lower level decision makers on the final decisions. Such a decision problem is called as a bilevel decision problem. The decision maker at the upper level is known as the leader, and at the lower level, the follower.

Bilevel decision problems have been introduced by Von Stackelberg in the context of unbalanced economic markets in the fifties of the 20th century [Stackelberg 1952]. After that moment a rapid development and intensive investigation of these problems begun both in theoretical and in applications oriented directions [Chen and Gruz 1972] [Candler and Norton 1977] [Bialas and Townsley 1982] [Bard and Falk 1982] [Bard and Moore 1992] [Bard 1998] [Dempe 2002]. Contributions to this field have been delivered by mathematicians, economists and engineers and the number of papers within this field is ever growing rapidly. This interest stems from the inherent complexity and consequent challenge of the underlying mathematics, as well as the applicability of the bilevel decision model to many real-world situations.

From its inception, bilevel decision problems have been introduced to the optimization community. Most of the efforts concentrated on theoretical or applied development for the linear or nonlinear version of the problem, such as K-Best approach [Bialas and Karwan 1984] or Kuhn-Tucker approach [Bard and Falk 1982] for solving linear bilevel programming problems, and Penalty function approach [White and Anandalingam 1993] or stability based approach [Liang and Sheng 1992] for solving nonlinear bilevel programming problems. However, bilevel decision making may involve many uncertain factors in a real world problem. Therefore it is hard to determine the objective functions and constraints when build a bilevel decision model. In addition, even if all the functions are linear, the resultant model may be difficult to be solved by the methods of optimization [Bard 1998]. To handle the two issues, it therefore needs to explore establishing a bilevel model by using uncertain information processing techniques.

Our previous work presented a new definition of solution and related theorem for linear bilevel programming, thus solved a fundamental deficiency of existing linear bilevel programming theory [Shi et al 2005]. We also developed an extended Kuhn-Tucker approach [Shi et al 2005a] and an extended *K*th-best approach [Shi et al 2005b] for solving linear bilevel decision problems. As a new exploration to model and solve a bilevel decision problem, this paper first formulates a bilevel decision problem using decision rule sets. It then applies the methods of rough set to reduce the models. With the methods of value reduction in rough set theory, simpler decision rule sets are extracted from decision rule sets (decision tables) to represent the evaluation methods of the objectives or the constraints. Besides, attribute importance degree based rule trees are used to solve uncertain problems and get the final decision. The structures can be extended beyond two levels with the realization that attending behavioral and operational relations become much more difficult to conceptualize and describe. The paper is divided into five sections. Section 2 introduces the preliminaries of this paper, including some notions about decision tables and decision rules. Section 3 proposes the model of the rule sets based bilevel decision problem, and then the algorithm of modelling. In Section 4, an example is presented. The last section includes the conclusion and future work.

2 Preliminary

For the convenience of description, we introduce some basic notions of decision tables and decision rules. Besides, we also develop some related definitions that will be useful in this paper.

2.1 Decision Tables

A decision table is commonly viewed as a functional description, which maps inputs (conditions) to outputs (actions) without necessarily specifying the manner in which the mapping is to be implemented [Lew and Tamanaha 1976]. The formal definition is as follows.

Definition 2.1[Wang 2001] **(Decision tables):** A decision table is defined as

$$S=<U, R, V, f>,$$

where U is a finite set of objects; $R=C\cup D$ is a finite set of attributes, C is the condition attribute set and D is the decision attribute set; set of its values V_a is associated for every attribute $a\in R$, and $V=\cup_{r\in R}V_r$; and each attribute has a determine function f: $U\times R\rightarrow V$, and f determines the attribute value of each object x.

A decision table is as a special and important knowledge expression system. It shows that, when some conditions are satisfied, decisions, actions, operations or controls can be made. Decision attributes in a decision table can be unique or not. In the latter case, the decision table can be converted to one with unique decision attribute [Wang 2001]. So, we suppose that there is only one decision attribute in decision tables in this paper.

2.2 Decision Rules

Definition 2.2[Wang 2001] **(Decision rules):** Let $S = <U, R, V, f>$ be a decision table, and $B\subseteq C$. Then a decision rule dr is generated from B and D with the form

$$dr: \wedge\{(a, v_a)\}\Rightarrow (d, v_d),$$

where $a\in B$, $v_a\in V_a$, $d\in D$, $v_d\in V_d$, and V_a, V_d is defined by Def. 2.1; $\wedge\{(a, v_a)\}$ is called as the precondition of a decision rule (denoted as Con_{dr}) and (a, v_a) is called as an element in the precondition; (d, v_d) is called as the decision of a decision rule (denoted as Des_{dr}).

It is obvious that objects in decision tables can be expressed by decision rules.

In order to describe the rule sets based bilevel decision model clearly, we present some notions related with decision rules as follows.

Definition 2.3 (Father decision rules): Decision rule dr_1 is said to be the father rule of decision rule dr_2, if each element in Con_{dr1} is also an element in Con_{dr2} and there is at least one element in Con_{dr2} that is not an element in Con_{dr1}. Here, dr_2 is said to be the son rule of dr_1.

Definition 2.4 (Objects which are consistent with a decision rule): A object o is said to be consistent with decision rule dr: $\wedge\{(a, v_a)\}\Rightarrow(d, v_d)$ $(a\in B, d\in D)$, if for $\forall a\in(B\cup D)$, $o_a=v_a$ is satisfied, where o_a is the value of o on attribute a. Given a decision table S, the set of all objects in S that are consistent with decision rule dr is denoted as $[dr]_S$.

Definition 2.5 (Objects which are conflict with a decision rule): A object o is said to be conflict with decision rule dr: $\wedge\{(a, v_a)\}\Rightarrow(d, v_d)$ $(a\in B, d\in D)$, if for $\forall a\in B$, we have $o_a=v_a$, and $o_d\neq v_d$. Given a decision table S, the set of all objects in S that are conflict with decision rule dr is denoted as $[\cancel{dr}]_S$.

Definition 2.6 (Rules which are consistent with a decision table): A decision rule dr is said to be consistent with a decision table S, if there isn't any object in S that is conflict with dr.

Definition 2.7 (Rule inclusion): Decision rule dr_1 is said to be including decision rule dr_2, if all objects which are consistent with dr_2 are also consistent with dr_1, denoted as Incl(dr_1, dr_2). In this case, if the number of the elements in dr_1's precondition is the same as that in dr_2's precondition, then dr_1 is said to be equal to dr_2.

Definition 2.8 (Rule conflict): Decision rule dr_1 is said to be conflict with decision rule dr_2, if all objects satisfied dr_2 are conflict with dr_1, which is denoted as Conf(dr_1, dr_2). In this case, if the number of the elements in dr_1's precondition is the same as that in dr_2's precondition, then dr_1 is said to be completely conflict with decision rule dr_2, else dr_1 is said to be partly conflict with dr_2.

Definition 2.9 (Rule length): Rule length is the number of elements in the rule's precondition.

Decision rule set RS is the set of decision rules. It can be divided into the following two categories (Def 2.10 and Def. 2.11) according to whether there are conflicts among its rules.

Definition 2.10 (Consistent decision rule sets): A decision rule set RS is said to be consistent, if there isn't any rule in the rule set conflicting with other rules in the rule set, that is to say, $\forall dr_1, dr_2\in RS$ ($\neg$ Conf(dr_1, dr_2)).

Definition 2.11 (Inconsistent decision rule sets): An decision rule set RS is said to be inconsistent, if there is some rule in the rule set conflicting with at least one another rule in the rule set, that is to say, $\exists\, rule_1 \in RS$ ($\exists\, rule_2 \in RS$ (Conf($rule_1$, $rule_2$))).

Definition 2.12 (Simplest decision rule sets): Suppose *dr* is a random decision rule in a consistent decision rule set *RS*, if *dr* is replaced by one of its father rules *fdr* and the resultant decision rule set is still consistent, then *RS* is said to be a redundant decision rule set, otherwise, it is said to be a simplest decision rule set.

3 Rule Sets Based Bilevel Decision Problem Modelling

When solving a bilevel decision problem, which objective functions and constraints related are expressed by linear or nonlinear functions, optimization approaches can be used. However, some real-world problems can't be easily formulated or approximated as linear or nonlinear programs. To handle the issue, new models for bilevel decision problems are needed.

A decision table can be used to lay out in a tabular form all possible situations where a decision may encounter and to specify which action to take in each of these situations. They can be used in projects to clarify complex decision making situations. Decision tables are commonly thought to be restricted in applicability to procedures involving sequencing of tests, nested-IFs, or CASE statements. In fact, a decision table can implement any computable function. It was observed that any Turing Machine program can be "emulated" by a decision table by letting each Turing Machine instruction of the form (input, state) + (output, tape movement, state) be represented by a decision rule (or an object in a decision table) where (input, state) are conditions and (output, tape movement, state) are actions. From a more practical point of view, it can also be shown that all computer program flowcharts can be emulated by decision tables [Lew and Tamanaha 1976].

Therefore, in theory, after emulating all possible situations in a domain, constraints of a decision problem can be transformed to a decision table, named as a constraint decision table. In a similar way, objective functions can also be transformed to a decision table, named as objective decision table. That is to say, a bilevel decision problem can be transformed into a set of decision tables, where decision variables are represented by the objects in these decision tables.

Rule sets are more general knowledge generated from decision table and they had stronger knowledge expressing ability than decision table. Rule sets overcome the following disadvantages of decision tables:

1) For complex situations, decision tables may become extremely large;

2) The objects in decision tables lack of adaptability. They can't adapt any new situations and one object can only record a situation.

So, we use rule sets to describe the objectives and constraints. The bilevel decision problem, which objectives and constraints of both leader and follower are described by rule sets, is called as a rule sets based bilevel decision model. And the bilevel decision model based on decision tables is a special case of rule sets based decision model.

3.1 Decision Rule Set Function

To present the model of rule sets based bilevel decision model, the definition of decision rule set function is needed.

Given a decision table $S=<U, R, V, f>$, where $R=C\cup D$ and $D=\{d\}$. Suppose x and y are two variables, where $x\in X$ and $X=V_{a1}\times\ldots\times V_{am}$, $y\in Y$ and $Y=V_d$. V_r is the set of attribute r's values and $a_i\in C$, i=1 to m and m is the number of condition attributes. *RS* is a decision rule set generated from *S*.

Definition 3.1 (Decision rule set function): A decision rule set function *rs* from *X* to *Y* is a subset of the cartesian product $X \times Y$, such that for each *x* in *X*, there is a unique *y* in *Y* generated with *RS* such that the ordered pair (x, y) is in *rs*. *RS* is called as the decision rule set related with the function, *x* is called as the condition variable, *y* is called as the decision variable, *X* is the definitional domain and *Y* is the value domain.

Calculating the value of a decision rule set function is to make decisions for undecided objects with decision rule sets, where undecided objects are objects without decision values. In order to present the method of calculating the value of a decision rule set function, we first introduce a definition.

Definition 3.2 (Undecided objects matching a decision rule): An undecided object o is said to be matching a decision rule dr: $\wedge\{(a, v_a)\}\Rightarrow(d, v_d)$, where $a\in B$, $d\in D$, if for each $a\in B$, $o_a=v_a$ is satisfied, where o_a is object o's value on attribute a.

Given a decision rule set *RS*, all rules in *RS* that is matched by object o is denoted as MR_{RS}^{o}.

With the definitions, a brief method of calculating the result of a decision rule set function is showed as follows:

Step 1: Calculate MR_{RS}^{o};

Step 2: Select a decision rule dr from MR_{RS}^{o}, where $dr:\wedge\{(a, v_a)\}\Rightarrow(d, v_d)$;

Step 3: The value of $rs(o)$ is set to be v_d, that is, $rs(o)=v_d$.

Complete

It is obvious that there may be more than one rule in MR_{RS}^{o}. In this case, when decision values of the rules in MR_{RS}^{o} are different, the result could be various according to above method, which is called as the uncertainty in a decision rule set function. Methods of selecting the final rule from MR_{RS}^{o} are very important, and they are said to be the uncertainty solution methods.

The elimination of uncertainty is a process of selection. We can select a rule rightly only when some information is known. In other words, we are said to be informed only when we can select rightly and definitely. In this paper, we present a rule tree based model to deal with the uncertainty in Section 3.2.

3.2 Rule Trees

Rule tree is a compact and efficient structure expressing a rule set. We first introduce the definition of rule tree, which is developed in our previous work [Zheng and Wang 2004]. Based on the definition of rule tree, we improve the rule tree structure with two constraints.

Definition 3.3 (Attribute importance degree based rule tree): Attribute importance degree based rule tree is a rule tree, and it satisfies the following two conditions:

1) The attribute expressed by the upper level is more important than that expressed by any lower level;

2) Among the branches with the same start node, the value represented by the left branch is more (or better) than the value represented by any right branch. And every possible value is more (better) than the value "*".

Definition 3.4 (Comparison of rules): Rule dr_1: $\wedge\{(a_i, v_{a1i})\} \Rightarrow (d_1, v_{d1})$ is better than rule dr_2: $\wedge\{(a_i, v_{a2i})\} \Rightarrow (d_2, v_{d2})$, if v_{a1k} is better than v_{a2k} or the value of a_k is deleted from rule dr_2, where attribute a_i is more important than a_{i+1}, and for each $j<k$, $v_{a1j}=v_{a2j}$.

Theorem 3.1: The rule expressed by the lefter branch in an attribute importance degree based rule tree is better than the rule expressed by the righter branch.

It is obvious that the theorem holds from Def. 3.4.

Theorem 3.2: After transformed to an attribute importance degree based rule tree, the rules in a rule set are total order, that is to say, every two rules can be compared.

It is obvious that the theorem holds from Def. 3.4 and Theorem 3.1.

3.3 Rule Sets Based Bilevel Decision Model

In the following, the mathematical model of rule sets based bilevel decision model is presented. Here, we suppose there are one leader and one follower. Besides, we suppose that, if x is the undecided object of the leader and y is the undecided object of the follower, then $x \oplus y$ is the combined undecided object of the leader and the follower together.

Definition 3.5 (Model of rule sets based bilevel decision):

$$\begin{aligned}
&\min_{x}\ f_L(x \oplus y)\\
&\quad \text{s.t. } g_L(x \oplus y) \geq 0\\
&\quad\ \min_{y}\ f_F(x \oplus y)\\
&\qquad \text{s.t. } g_F(x \oplus y) \geq 0, \qquad (3.1)
\end{aligned}$$

where x and y are undecided objects of the leader and the follower respectively. f_L and g_L are the objective decision rule set function and constraint decision rule set function of the leader respectively, f_F and g_F are the objective decision rule set function and constraint decision rule set function of the follower respectively. F_L, G_L, F_F and G_F are the corresponding decision rule sets of above decision rule set functions respectively.

3.4 Modelling Algorithm of Rule Sets Based Bilevel Decision Model

In the following, we present the modelling algorithm of rule sets based bilevel decision model.

Algorithm 3.1 (Modelling Algorithm of Rule Sets Based Bilevel Decision Model)

Input: A bilevel decision problem with its objectives and constraints of both the leader and the follower;

Output: A rule sets based bilevel decision model;

Step 1: Transform the problem with decision rule sets;

Step 2: Preprocess F_L, such as delete reduplicate rules from the rule sets, eliminate noisy and etc.;

Step 3: If F_L need to be reduced,

then using reduction algorithm to reduce F_L;

Step 4: Preprocess G_L, such as delete reduplicate rules from the rule sets, eliminate noisy and etc.;

Step 5: If G_L need to be reduced,

then using reduction algorithm to reduce G_L;

Step 6: Preprocess F_F, such as delete reduplicate rules from the rule sets, eliminate noisy and etc.;

Step 7: If F_F need to be reduced,

then using reduction algorithm to reduce F_F;

Step 8: Preprocess G_F, such as delete reduplicate rules from the rule sets, eliminate noisy and etc.;

Step 9: If G_F need to be reduced,

then using reduction algorithm to reduce G_F;

Complete

Step 1 is the key step of the modeling process. The users can complete the step by lay out all possible situations, that is, transform the problem to decision tables. When the users know the general knowledge (rules) under the problem, they can directly transform the problem to some simpler decision rule sets. In general, the realization of the step depends on the characters of the problem and the users' knowledge related with the problem.

In Step 2, Step 4 and Step 6, the four rule sets are preprocessed. The process is very important, because incomplete, noisy and inconsistency are the common characters of huge and real data. So, we should use some techniques to eliminate these problems in data before modeling. In [Han and Kamber 2001], the issue is discussed in detail.

In Step 5, Step 7, and Step 9 of Alg. 3.1, rule set is reduced by some reduction algorithm. To reduce a decision rule set or extract decision rules from a decision table, the methods based on rough set theory are popular and efficient. Many rough set based decision rule extraction algorithms, named as value reduction algorithms, are developed in rough set theory [Pawlak 1991] [Hu and Cercone 1995] [Mollestad and Skowron 1996] [Wang 2001] [Zheng and Wang 2004]. And the algorithms made successful applications in many fields [Kiak 2001] [Pawlak and Slowinski 1986] [Kiak 2001] [Carlin et al 1998]. Besides, there are some rough set based systems, such as ROSETTA [ROSETTA], RIDAS [Wang et al 2002], RSES [Jan et al 2002] and so on, can be used to extract decision rule sets from decision tables. So, we use rough set theory based methods to reduce the rule sets based models in this paper.

Based on rough set theory, various value reduction algorithms can be developed. Value reduction is a process to find a subset of values in decision rule set which satisfies that removing any value in this subset will definitely cause new inconsistency. There are many value reduction algorithms [Wang 2001] [Hu and Cercone 1995] [Mollestad and Skowron 1996] [Zheng and Wang 2004]. A simplest decision rule set (Def. 2.12) can be extracted from a rule set or decision table with the reduction algorithms of rough sets.

In the following section, we use an example to illustrate the modelling process.

4 Example

Suppose there is a factory with two levels in its staff management. The upper level is the factory executive committee and the lower is a workshop management committee. Now, the factory wants to recruit new workers. The factory executive committee should consider the overall objectives, and the workshop management committee considers its own needs, so the objectives for the two levels may be different. The executive committee of factory could ask the workshop to calculate and submit an optimal production plan as though it were operating in isolation. Once the plans are submitted, they are modified with the overall objective of the factory in mind. An output plan ultimately emerges that is optimal for the factory as a whole.

When decide whether a person could be recruited, the factory executive committee considers the following two factors, which are team spirit and organizational ability of the person; and the workshop management committee considers two factors, which are age and eyesight of the person. Suppose the condition attributes in ascending order according to the importance degree are "Team Spirit", "Organizational Ability", "Age", "Eyesight".

The two committees can't express the conditions of the workers they want recruit to linear or nonlinear functions. But they have a base recorded the worker's information having been recruited. So, we can transform the base to two decision tables (Table 4.1, 4.2), which are the objective rule sets of the leader and the follower. The objects of the decision tables represent workers. The condition attributes of the decision tables are the factors; the decision attributes of the two decision tables are both the accept grade of the worker represented by the condition attribute values. The constraints of the two committees are expressed by simple rule sets (Equation 4.1, 4.2), which define the constraint region.

Then, we use Alg. 3.1 to transform the problem to rule sets based bilevel model.

Alg. 3.1-Step 1: Transform the problem with decision rule sets. Table 4.1 represents the objective rule set of the leader, Table 4.2 represents the objective rule set of the follower, Equation 4.1 represents the constraint rule set of the leader and Equation 4.2 represents the constraint rule set of the follower.

Table 4.1 Objective rule set of the leader

Team Spirit	Organizational Ability	Age	Eyesight	Accept Grade
Poor	Middle	Middle	Middle	2
Good	Middle	Middle	Middle	1
Good	Fine	Old	Middle	1
Middle	Poor	Young	Poor	3
Poor	Poor	Middle	Middle	3
Middle	Poor	Old	Poor	3

Good	Middle	Middle	Good	1
Good	Fine	Middle	Middle	1
Middle	Fine	Old	Poor	2
Good	Fine	Old	Good	1
Good	Poor	Old	Good	3
Good	Fine	Young	Good	1
Good	Poor	Young	Middle	3

The constraint rule set of the leader:

G_L= { (Team Spirit, Good) $\Rightarrow$ (*pc*, 1)

(Team Spirit, Middle) $\Rightarrow$ (*pc*, 1)

} (4.1)

Table 4.2 Objective decision table of the follower

Organizational Ability	Age	Eyesight	Accept Grade
Fine	Young	Poor	2
Poor	Old	Good	2
Fine	Young	Good	1
Fine	Old	Middle	1
Poor	Young	Middle	3
Middle	Middle	Poor	2
Poor	Middle	Poor	3
Poor	Old	Poor	3
Fine	Old	Good	1
Poor	Young	Good	2
Middle	Young	Middle	2
Poor	Middle	Good	2
Fine	Old	Good	1
Middle	Middle	Good	2
Fine	Middle	Poor	2

The constraint rule set of the follower:

G_F= {(Eyesight, Poor) $\Rightarrow$ (*pc*, 0)} (4.2)

Because the scale of the data is very small, the preprocess steps(Step 2, Step 4, Step 6 and Step 8) are not needed. Besides, the constraint rule sets of the leader and the follower are very brief, so the reduction steps of G_L and G_F (Step 5 and Step 9) are not needed.

In the constraint rule sets, we suppose that, if the decision of a rule is (*pc*, 0), any undecided objects consistent with the rule are not in the constraint region; if the decision value of a rule is (*pc*, 1), any undecided objects consistent with the rule are in the constraint region. We can also use some other formats of the constraint rule to express the constraint region.

Alg. 3.1-Step 3 and Step 7: Reduce the objective rule sets of the leader and the follower.

After reducing the decision tables based on rough set theory, we can get reduced objective rule sets of the leader and the follower (4.3, 4.4). Here, we use the decision matrices based value reduction algorithm [Ziarko et al 1996] in RIDAS system [Wang et al 2002].

The reduced objective rule set of the leader:

F_L={(Team Spirit, Poor) $\wedge$ (Organizational Ability, Middle) $\Rightarrow$ (Accept Grade, 2)

(Team Spirit, Good) $\wedge$ (Age, Middle) $\Rightarrow$ (Accept Grade, 1)

(Team Spirit, Good) $\wedge$ (Organizational Ability, Fine) $\Rightarrow$ (Accept Grade, 1)

(Organizational Ability, Poor) $\Rightarrow$ (Accept Grade, 3)

(Team Spirit, Middle) $\wedge$ (Organizational Ability, Fine) $\Rightarrow$ (Accept Grade, 2)

} (4.3)

The reduced objective rule set of the follower:

F_F={(Organizational Ability, Fine) $\wedge$ (Eyesight, Poor) $\Rightarrow$ (Accept Grade, 2)

(Organizational Ability, Poor) $\wedge$ (Eyesight, Good) $\Rightarrow$ (Accept Grade, 2)

(Organizational Ability, Fine) $\wedge$ (Eyesight, Good) $\Rightarrow$ (Accept Grade, 1)

(Organizational Ability, Fine) $\wedge$ (Age, Old) $\Rightarrow$ (Accept Grade, 1)

(Organizational Ability, Poor) $\wedge$ (Eyesight, Middle) $\Rightarrow$ (Accept Grade, 3)

(Organizational Ability, Middle) $\Rightarrow$ (Accept Grade, 2)

(Organizational Ability, Poor) $\wedge$ (Eyesight, Poor) $\Rightarrow$ (Accept Grade, 3)

} (4.4)

With above steps, we get the rule sets based bilevel decision model of the problem, that is

$$\begin{aligned} &\min_x \; f_L(x \oplus y) \\ &\quad \text{s.t. } g_L(x \oplus y) \geq 0 \\ &\quad \min_y \; f_F(x \oplus y) \qquad (4.5) \\ &\quad\quad \text{s.t. } g_F(x \oplus y) \geq 0, \end{aligned}$$

where f_L, f_F, g_L, g_F are the corresponding decision rule set functions of F_L, F_F, G_L, G_F.

5 Conclusion and Future Work

In this paper, we explore the use of rule set approach to format a bilevel decision problem by establishing a rule sets based model. We have seen that the common features of bilevel decision problems are:

a) Interactive decision making units exist within a predominantly hierarchical structure;

b) The lower level executes its policies after, and in view of, decisions made at the upper level;

c) Each unit independently maximizes net benefits (minimizes net costs), but is affected by the actions of other units through externalities;

d) Extramural effects enter a decision maker's problem through his objective function and feasible strategy set.

Rule sets based bilevel decision problems incorporate above features. From these features, it is obvious that to solve a rule sets based bilevel problem should be based on the solving method of rule sets based multiple objectives decision problems. Besides, we can divide the algorithms solving rule sets based decision problems into three categories, that is, forward algorithms, reverse algorithms and mixed algorithms. These issues would be discussed in our future work.

6 Acknowledgments

The work presented in this paper was supported by Australian Research Council (ARC) under discovery grants DP0557154 and DP0559213, and University of Technology, Sydney (UTS) under grant

7 References

Bard, J.F. (1998), *Practical Bilevel Optimization: Algorithms and Applications*, Kluwer Academic Publishers, USA.

Bard, J.F. and Falk, J.E. (1982), An Explicit Solution to the Multi-Level Programming Problem, *Computers & Operations Research* **9**, 77-100.

Bard, J.F. and Moore, J.T. (1992), An Algorithm for the Discrete Bilevel Programming Problem, *Naval Research Logistics* **39,** 419-435.

Bialas, W. F. and Karwan, M. H. (1982), On Two-Level Optimization, *IEEE Trans Automatic Control* **AC-26,** 211-214.

Bialas, W. and Karwan, M. (1984), Two-Level Linear Programming, *Management Science* **30,** 1004–1020.

Candler, W. and Norton, R. (1977), *Multilevel Programming and Development Policy*, World Band Staff Work No. 258, IBRD, Washington, D.C..

Carlin, U. S., Komorowski, J. and Ohrn, A. (1998), Rough set analysis of patients with suspected of acute appendicitis, *Proc. IPMU'98*, Paris, France, 1528–1533.

Chen, C.I. and Gruz, J.B. (1972), Stackelberg Solution for Two Person Games with Biased Informtaion Patterns, *IEEE Trans. On Automatic Control* **AC-17,** 791-798.

Dempe, S. (2002), *Foundations of Bilevel Programming*, Kluwere Academic Publishers.

Han, J. and Kamber, M. (2001), *Data Mining Concepts and Techniques*, Morgan Kaufmann Publishers.

Hu, X.H. and Cercone, N. (1995), Learning in relational database: a rough set approach, *Computational Intelligence*, **11**, 323-338.

Kiak, A. (2001), Rough Set Theory: A Data Mining Tool For Semiconductor Manufacturing, *IEEE Transaction on Electronics Packaging Manufacturing*, **24**, 44-50.

Jan, G. B., Marcin, S. S. and Jakub, W. (2002), A New Version of Rough Set Exploration System, *Rough Sets and Current Trends in Computing,* Publisher? 397-404.

Lew, A. and Tamanaha, D. (1976), Decision table programming and reliability, *Proc. 2nd Intl. Conf. Software Engineering,* San Francisco, 345-349.

Liang, L. and Sheng S.H. (1992), The Stability Analysis of Bilevel Decision and Its Application, *Decision And Decision Support System* **2**, 63-70.

Mollestad, T. and Skowron, A. (1996), A rough set framework for data mining of propositional default rules. *Proc. Foundations of Intelligent Systems of the 9th International Symposium,* 448-457, Springer-Verlag.

Pawlak, Z. (1991), *Rough sets Theoretical Aspects of Reasoning about Data*, Boston, Kluwer Academic Publishers.

Pawlak, Z. and Slowinski, K. (1986), Rough Classification of Patients After Highly Selective Vagotomy Duodenal Ulcer, *International Journal of Man-Machine Studies* **24**, 413-433.

Pooch, U.W. (1974), Translation of decision tables, *ACM Computing Survey* **6**, 125-151.

ROSETTA: The ROSETTA Homepage, http://www.rosettaproject.org/.

Schlimmer, J. C. and Fisher, D A. (1986), Case study of incremental concept induction, *Proceedingsof the Fifth National Conf. on Artificial Intelligence*, **1,** 496-501.

Shan, N., Ziarko, W., Hamilton, H. J., and Cercone, N. (1995), Using rough sets as tools for knowledge discovery, *Proc. 1st Int. Conf. Knowledge Discovery Data Mining*, Menlo Park, CA, 263–268.

Shi, C., Lu, J. and Zhang, G. (2005a), An extended Kuhn-Tucker approach for linear bilevel programming, *Applied Mathematics and Computation* **162,** 51-63.

Shi, C., Lu, J. and Zhang, G. (2005b), An extended *K*th-Best approach for linear bilevel programming, *Applied Mathematics and Computation* **164,** 843-855.

Shi, C., Zhang, G. and Lu, J. (2005), On the definition of linear bilevel programming solution, *Applied Mathematics and Computation* **160,** 169-176.

Stackelberg, H. V.(1952), *The Theory of Market Economy*, Oxford, Oxford University Press.

Skowron, A. and Polkowski, L. (1998), *Rough Sets in Knowledge Discovery*, Physica Verlag, Heidelberg.

Wang, G.Y. (2001), *Rough set theory and knowledge acquisition*, Press of Xi'an Jiaotong University (In Chinese).

Wang, G. Y., Zheng, Z. and Zhang, Y. (2002), RIDAS-A Rough Set Based Intelligent Data Analysis System, *Proceedings of the First International Conference on Machine Learning and Cybernetics*, 646~649.

White, D. and Anandalingam, G. (1993), A Penalty Function Approach For Solving Bi-Level Linear Programs, *Journal of Global Optimization* **3,** 397–419.

Zheng, Z. and Wang, G.Y. (2004), RRIA:A Rough Set and Rule Tree Based Incremental knowledge Acquisition Algorithm, *Fundamenta Informaticae* **59,** 299-313.

Ziarko, W., Cercone, N. and Hu, X. (1996): Rule Discovery from Databases with Decision Matrices, *9th Int. Symposium on Foundation of Intelligent Systems,* 653-662.

CASO: A Framework for dealing with objectives in a constraint-based extension to AgentSpeak(L)

Aniruddha Dasgupta **Aditya K. Ghose**

Decision Systems Lab
School of IT and Computer Science
University of Wollongong,
Wollongong, NSW 2522,
Email: ad844@uow.edu.au, aditya@uow.edu.au

Abstract

Incorporating constraints into a reactive BDI agent programming language can lead to better expressive capabilities as well as more efficient computation (in some instances). More interestingly, the use of constraint-based representations can make it possible to deal with explicit agent objectives (as distinct from agent goals) that express the things that an agent may seek to optimize at any given point in time. In this paper we extend the preliminary work of Ooi et.al in augmenting the popular Belief-Desire-Intention (BDI) language AgentSpeak(L) with constraint-handling capabilities. We present a slightly modified version of their proposal, in the form of the language CAS (Constraint AgentSpeak). We then extend CAS to form the language CASO (Constraint AgentSpeak with Objectives) to incorporate explicit objectives (represented as objective functions) and present techniques for performing option selection (selecting the best plan to use to deal with the current event) as well as intention selection. In both cases, we present parametric look-ahead techniques, i.e., techniques where the extent of look-ahead style deliberation can be adjusted.

1 Introduction

The concept of using constraints has been introduced by Ooi *et al.* (1999) where it has been shown that the integration of constraints in a high-level agent specification language yields significant advantages in terms of both expressivity and efficiency. The BDI framework employed in the multi agent broker system is implemented with an improvised computation strategy - a synergy of unification and constraint solving. The improvisation applies constraint directed solving on the context section of a BDI agents plan specification in order to determine an application plan to fire. The constraint system introduced into the BDI framework maintains a constraint store that collects a set of constraints that augment the beliefs of an agent.
In this paper we extend the work one by Ooi *et al.* (1999) by incorporating explicit objectives beside the constraints. We also describe some efficient plan and intention selection methods which would result in better expressibility and more efficient computation which has not been addressed in either Agentspeak(L) introduced by Rao (1996) or by Ooi *et al.* (1999). This type of selection mechanisms are particulary useful in many real world applications which require the use of intelligent agents to perform some critical tasks. This paper extends the preliminary work presented by Dasgupta *et al.* (2005).
The remainder of this article is organized as follows. Section 2 gives an example which is used throughout the rest of the paper. Section 3 introduces the language CASO and section 4 discusses its operational semantics and describes the algorithms for efficient plan and intention selection. Finally, concluding remarks and comparisons are presented in the last section.

2 Motivation

In this section we give an example of detailed reasoning behind the adoption of CASO. We begin by outlining a specific scenario of using CASO by a truck in order to deliver goods one location to another. The roads that the truck would take consists of several roads with choices available at various important points to follow one of the many paths. For simplicity, let us assume that the truck can either take the city road or the highway and both runs in parallel and the truck can at exit from the highway into a city road or enter the highway from city road from the important points.

Let us assume that there following tasks that need to be achieved.
G1. Deliver a parcel X to location B from the current location A.
G2. Fill up the tank whenever there is less than a quarter of petrol in the tank.

The following objectives may also be supplied to the truck driver.
O1. Choose the shortest path for delivery of the parcel.
O2. Minimize the amount of petrol required.

A constraint the truck driver may be supplied with might be the following.
C1. Parcel must be delivered by 5p.m.

Let us also assume the following ground beliefs.
B1. Petrol consumption rate in highways is 10 k.m./litre.
B2. Petrol consumption rate in city roads is 8 k.m./litre.

The two goals above are fairly independent of each other. Within an agent context the above tasks may be represented as a set of goals that need to be fulfilled. In order to fulfil each goal, the truck driver needs to execute a sequence of actions (i.e. to execute a plan). There might be a number of plans for

achieving the same task. As an example, for achieving the first goal there might be two possible plans:
Plan P1: *1. From location A take H1. 2. Deliver the parcel X at B.*
Plan P2: *1. From location A take city road R1. 2. Deliver the parcel X at B.*
Note that each of the plans above may have subplans which would describe the exact route to be followed. Both the above plans achieve the same result of delivering the parcel. However, the difference that exist are the time and petrol needed. In case of plan P1, the time taken is less as there is less traffic and for plan P2, the amount of fuel required is less whereas time taken is more.

3 Agent Programming with CASO

Informally, an agent program in CASO consists of a set of beliefs B, a set of constraints C, an objective function O, a set of events E, a set of intention I, a plan library P, a constraint store CS, an objective store OS and three selection functions S_E, S_P, S_I to select an event , a plan and an intention respectively to process and n_p and n_i are the two parameters which denote the number of steps to look-ahead for plan and intentions selection respectively.

Definition 1:
An agent program is a tuple {B, P,E, I,C,O,S_O, S_E, S_I, n_p, n_i,CS,OS} where
B is a set of Beliefs.
P is agent plan repository, a library of agent plans.
E is set of events (including external and internal).
I is a set of intentions.
C is a set of constraints.
O is an objective function.
S_E is a selection function which selects an event to process from set E of events.
S_O is a selection function which selects an applicable plan to a trigger t from set P of plans.
S_I is a selection function which selects an intention to execute from set I of intentions.
CS is a constraint store which stores constraints which come as events.
OS is an objective store which stores the objective function which comes as an event.
n_p is an integer which denotes the number of steps required to look-ahead for plan selection.
n_i is an integer which denotes the number of steps required to look-ahead for intention selection.

In CASO, a constraint directed improvisation is incorporated into the computation strategy employed during the interpretation process. Constraint logic programming (CLP) combines the flexibility of logic with the power of search to provide high-level constructs for solving computationally hard problems such as resource allocation.
Formally, a language CLP(X) is defined by *a constraint domain X, a solver for the constraint domain X and a simplifier for the constraint domain X.*

Definition 2:
A CASO plan p is of the form $t : b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_m \leftarrow sg_1, sg_2, \cdots, sg_k$ where t is the trigger; each b_i refers to a belief; each c_i is an atomic constraint; each s_g is either an atomic action or a subgoal.
For brevity we will use *BContext(p)* to denote the belief context of plan.
Thus *BContext(p)* $\equiv b_1 \wedge b_2 \cdots \wedge b_n$
Similarly, we will use CContext(p) to denote the constraint context of plan p.
Thus *CContext(p)* $\equiv c_1 \wedge c_2 \cdots \wedge c_m$

In our trucking example the beliefs and plans could be given as follows where TF refers to 'Tank Full', FC to 'Full Capacity of tank' and CL to 'Current Level' :

Beliefs
TF = false.
FC = 60.

Plans
+!fill-tank(CL):
TF = false&FC = 60&CL < 0.25 × FC ← (stop-to-fill(gas-station)); delay(5).

The above plan simply states that in order to achieve the goal of filling the tank, the tank has to be quarter full and the actions to be taken would to stop at a gas station and fill up the tank and this would have a delay of Transition of agent program to process events depends on the event triggers. An event trigger, t, can be addition(+) or removal(-) of an achievement goal($\pm g_i$) or a belief ($\pm b_i$).

4 Operational Semantics of CASO

The CASO interpreter manages a set of events, a constraint store, a objective store and a set of intentions with three selection functions. Intentions are particular courses of actions to which an agent has committed in order to handle certain events. Each intention is a stack of partially instantiated plans. Events, which may start off the execution of plans that have relevant triggering events, can be external when originating from perception of the agents environment (i.e., addition and deletion of beliefs based on perception are external events) ; or internal, when generated form the agents own execution of a plan (i.e., as subgoal in a plan generates an event of the type addition of an achievement goal).
In the latter case, the event is accompanied with the intention which generated it (as the plan chosen for that event will be pushed on top of that intention). External events create new intentions, representing separated focuses of attention for the agents acting on the environment.
The constraint store is initialized by the relevant constraints whenever a trigger contains a constraint in its context. At every cycle of the interpreter, the constraint store is enhanced with new constraints when applicable selected plan is executed. These incremental constraints collecting process eventually leads to a final consistent constraints set. Constraint solving is applied to the context of each plan to determine applicable plans as well as to generate solutions for subsequent actions. Similarly, the objective store contains the set of objective functions that need to be maximized (or minimized) which are part of the event context and is similarly updated at each cycle.
In the following sections we explain the basics of how CASO interpreter works. At every interpretation cycle of an agent program, CASO updates a list of events, which may be generated from perception of the environment, or from the execution of intentions (when subgoals are specified in the body of plans). It is assumed that beliefs are updated from perception and whenever there are changes in the agents beliefs, this implies the insertion of an event in the set of events.

4.1 Plan selection

After S_E has selected an event, CASO has to unify that event with triggering events in the heads of plans. This generates a set of all relevant plans. The constraints (if any) that are included in the constraint part of the context are put in the constraint store. The context part of the plans is unified against the agents beliefs. Constraint solving is now performed on these relevant plans to determine whether the constraint(s) in the context of the plan is (are) consistent with the constraints already collected in the constraint store . This results in a set of applicable plans(plans that can actually be used at that moment for handling the chosen event).
The objective store maintains a set of objective function which may be present in the event context. At each interpreter cycle, the objective store is also updated with an objective function for maximizing (or minimizing).

Definition 3:
Given plans p1 and p2 in the plan library, and given a current constraint store C and a current objective store O, $p_1 \leq_{opt} p_2$ *if and only if:* $OptSol(C \cup CContext(p_1), OS) \geq OptSol(C \cup CContext(p_2), O)$ *where OptSol(Constraints,Objective) denotes the value of the objective function when applied to the optimal solution to the problem denoted by the pair (Constraints, Objective).*
We assume of course that $C \cup CContext(p_1)$ and $C \cup CContext(p_2)$ are solvable.

Optimization techniques are then applied by the optimizer to each of the applicable plan to determine an optimal solution. In effect we are solving a 'Constraint Satisfaction Optimisation Problem' (CSOP) which consists of a standard 'Constraint Satisfaction Problem' (CSP) and an optimisation function that maps every solution (complete labelling of variables) to a numerical value. S_O now chooses this optimal solution from that set, which becomes the intended means for handling that event, and either pushes that plan on the top of an existing intention (if the event was an internal one), or creates a new intention in the set of intentions (if the event was external, i.e., generated from perception of the environment). Thus plan selection is defined as follows:

Definition 4:
Given a trigger t and a set of applicable plans AppPlans(t) for t, a plan $p \in AppPlans(t)$ *is referred to as an O-preferred plan if and only if:* $p \leq_{opt} p_i$ *for all* $p_i \in AppPlans(t)$.

The agent program is also responsible for making sure that the objective store is consistent at any point of time. During each cycle of the interpreter, new objectives are added into the objective store and hence a consistency checker is used to maintain consistency. Formally a consistent objective store is defined as below.

Definition 5:
Given an objective store OS and a new objective f, the result of augmenting OS with f, denoted by OS^*_f*, is defined as* $\gamma(MaxCons(OS \cup f))$ *where* γ *is a choice function and MaxCons(X) is the set of all* $x \subseteq X$ *such that*
1. x is consistent and
2. there exists no x' such that $x \subset x' \subseteq X$ *and x' is consistent.*
It is to be noted here that the triggering event can be the removal of an objective function also. The new objective store is now given by $\gamma(MaxCons(OS \cup \overline{O}) \cap OS$ where γ is the choice function, OS is the objective store and $\overline{O}$ is the negation of the objective O.

Selection of O-preferred plan can be further enhanced by using np the lookahead parameter form plan selection. In case np=0, no look-ahead is performed and maximizing the objective function on the set of applicable plans would result in an O-preferred plan as described earlier. However, if $n_p > 0$ then a look-ahead algorithm (similar to the one used for choosing the next move in a two-player game) is performed to select the O-preferred plan.
We assume that the agent is trying maximize its objective function and the environment may change in the worst possible way which would minimize the objective function. The goal of the agent would be to select a plan which would maximize the minimum value of the objective function resulting from the selection of plans which may occur due to the set of new possible events that may come from the environment.
We follow the definition of goal-plan tree given by Thangarajah (2004) to decompose the set of plans into a tree structure. In CASO, goals are achieved by executing plans and each goal has at least one plan, if not many, that can be used to satisfy the goal. Each plan can include sub-goals, but need not have any. The leaf nodes of the tree are plan-nodes with no children (i.e., no sub-goals).

Definition 6:
The relationship between a top level goal, its plans and subgoals defines a tree structure for each top-level goal, which is termed the goal-plan tree for that goal.

Each goal-plan tree consists of - a number of 'AND' nodes which are subgoals that must be executed sequentially for the goal to succeed; and a number of 'OR' nodes which are subgoals any one of which must be executed for the goal to succeed.

In our trucking example,there are two important criteria, which the user may want to satisfy:
1. the vehicle should go from the starting point to the destination point as fast as possible
2. the vehicle should go from the starting point to the chosen destination by maximum fuel saving.
The cost function for one length unit of a road R_i may look as: $C_u(R_i) = K \times F_u(R_i) + 1$ where $C_u(R_i)$ is the cost of one length unit (for example one meter) of the road $R_i, F_u(R_i)$ is fuel consumption for one length unit of the road R_i, and K denotes the degree of compromise (it must be a number equal or greater then zero). If K = 0 then the fuel consumption will be ignored and only the number of length units will be important the algorithm will find the shortest way to the destination. If the K parameter is a high number, the fuel saving will be very important for the optimization algorithm. The (global) cost of N used roads will then be the sum of N road costs: $TC = \sum(L(R_i) \times C_u(R_i))$. TC is the total cost of plan for the optimization algorithm and $L(R_i)$ is the used length of a road R_i.

Given a set of applicable plans, the truck agent would always try to achieve this objective at every decision step. However, there could be unforeseen road blocks and other situations which may result in the truck from changing its route at any of these decision points. This may result in the truck in spending more fuel than that what it would have used. Thus the strategy for the agent is to compute in advance the worst case scenario that may occur due to the change in the highly dynamic environment. Let us consider that the example of two applicable plans p1 and p1 each having one subgoal.

Plan p_1
$+!location(truck, D1, k) : location(truck, R1) \& k \geq 0 \leftarrow !follow(A, F1, L1, k)$

Plan p_2
$+!location(truck, D1, k) : location(truck, R1) \& k \geq 0 \leftarrow !follow(B, F2, L2, k)$

p_1 suggests that if truck is at location R1 and it needs to go to destination D1 then it can follow route A. p_2 suggests an alternate route of going to D1 from R1 given by B. F1 and F2 are the fuel consumption per kilometer of distance and L1 and L2 are the lengths of the two roads and k is the fuel compromise factor as described earlier. Let us assume that plan p_1 and p_2 have the following possible subplans.

Plan $p_{1.1}$
$+!follow(A, F, L, k) : F = 3 \& L = 3 \& (timeleft < 1) \& k \geq 0 \& k \leq 2 \leftarrow +!drive(A)$

Plan $p_{1.2}$
$+!follow(A, F, L, k) : F = 1.5 \& L = 3 \& (timeleft > 1) \& k \geq 2 \leftarrow +!drive(A)$

Plan $p_{2.1}$
$+!follow(B, F, L, k) : F = 3 \& L = 2 \& (timeleft < 1) \& k \geq 0 \leftarrow +!drive(B)$

Plan $p_{2.2}$
$+!follow(B, F, L, k) : F = 0.5 \& L = 2 \& (timeleft < 1) \& k \geq 2 \leftarrow +!drive(B)$

Plan $p_{1.1}$ suggest that if current time left to reach destination is less than 1 hr. then, the value of k should lie between 0 and 2. Similarly, plan $p_{1.2}$ suggests k should be greater than 2 if more than 1 hr. of time is left. Plan $p_{2.1}$ and $p_{2.2}$ suggest similar plans for route B.

Since the objective is to maximize the value of TC shown earlier, let the constraint solving yields the value of k=0 from plan $p_{1.1}$ and k=2 for plan $p_{1.2}$. Similarly, for plan p_2, the values for k are 0 and 2 respectively. Figure 1 shows the tree decomposition for plan p depicting all possible choices. The numbers corresponding to the leaf nodes are the values of the optimization function TC which we are trying to maximize. Thus choosing plan $p_{1.1}$ and $p_{1.2}$ would give values 3 and 12 respectively; similarly for plans $p_{2.1}$ and $p_{2.2}$ the corresponding values would be 2 and 4. Using the *LookAheadPlanSelection* algorithm shown below, we obtain the value of 3 at the root node which suggest that the agent should follow plan p_1.

Following the above algorithm, the truck agent would choose the p_1.

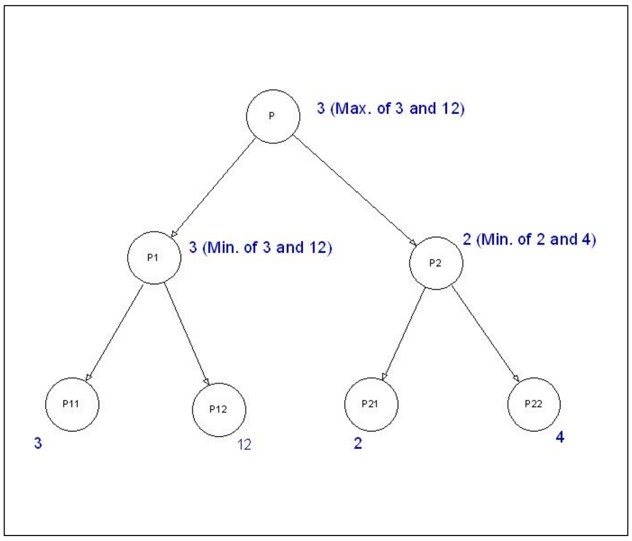


Figure 1: Plan Tree

Algorithm 1 LookAheadPlanSelection(int n, state S, ObjectiveStore OS, ConstraintStore CS)

```
1: Generate goal-plan tree up to n levels from current state S comprising of subgoals of AND and OR nodes with subplans.
2: Start from the root node.
3: Let constraint store at node $p = c_p$
4: Let $o_p$ denote the value of objective function at node p.
5: For each node p in the goal plan tree set $c_p \leftarrow CS$
6: **if** node p has child nodes $p_1, p_2 \cdots, p_k$ in an AND structure **then**
7:    Apply constraint solving at each $p_i$ with the current constraint store $cp_i$ and the set of constraints for $p_i$ to obtain $op_i$.
8:    Set $c_{pi+1} \leftarrow c_{pi}$ for all $i \geq 1$.
9:    Initialize constraint store for all child nodes of each $p_i$ with $c_{pi}$.
10: **end if**
11: **if** node p has child nodes $p_1, p_2 \cdots, p_k$ in an OR structure **then**
12:    Compute the objective function and update the constraint store for each $p_i$.
13:    Initialize constraint store for all child nodes of each $p_i$ with $c_{pi}$.
14: **end if**
15: **while** $n \neq 1$ **do**
16:    Propagate minimum value of objective function up to each parent node starting from the leaf node.
17:    $n = n - 1$
18: **end while**
19: Propagate the maximum value of its children for state S.
20: At state S, the best plan is the child with the maximum value.
```

4.1.1 Incremental Resolving of CSOPs

A single decision with such a strategy has $O(b^n)$ time complexity where b is the branching factor of the decision tree being explored and n is the number of steps to look ahead which is passed on as a parameter. The efficiency of plan selection can be greatly improved if we do not solve the CSOPs at every step from the beginning of n-step look-ahead at each decision point but instead apply some heuristic for incrementally resolving the CSOPs.

A heuristic similar to *Look Back* schemas like *backtracking* that are often used for consistency check in CSPs can be employed for resolving the CSOPs. Without any look-ahead, the CSOP for the given plan P is solved and the solution along with the set of constraints in the constraint store at that point is stored. In order to solve the CSOP for each step of look-ahead, the new set of constraints that are associated with each of the subplans at each decision point for plan P is added to the currently solved CSOP set -

if this new set of constraints violate the current value of the objective function, then backtracking is performed to the most recently instantiated variable that still has alternatives available and the new CSOP is solved with the new value of the instantiated variable.

4.2 Intention Selection and Execution

Once a plan is chosen the next stage is to execute a single intention in that cycle. The S_I function selects one of the agents intentions (i.e., one of the independent stacks of partially instantiated plans within the set of intentions). Look-ahead technique using decision tree is similarly employed here which could help in selecting an intention which would give the optimal solution. The parameter n_i denotes the number of steps for required to look ahead. In case of Intention selection, this merely becomes the number of items to be evaluated at the top of the intention stack. If there are more than one intention stacks present, then look-ahead procedure pops the top n_i elements of the stack from each intention and computes the optimal solution based on the constraint and the objective store.
Let us assume that in case of our truck agent, there are currently two intention stacks (I_1 and I_2) each corresponding to the two independent goals. The first one is to follow plan p_1 (described earlier) and the other one is to follow plan p_3 which describes the goal of picking up a parcel P2 from location C.

1. *Take route A to location D1 from* R1 *(plan p_1).*
2. *Pick up parcel P2 from location C (plan p_3).*

Both the above constitute a set of plans which are in the intention stack ready to be executed. The deliberation process of the truck agent is to decide which intention stack to pursue at a given point in time. For our example, let us assume that plan p_1 has subplan p_{11} in the intention stack (i.e. S_O has selected plan p_{11} to be executed). Let us also assume that the body of plan p_3 consists of the subplan p_{31} followed by subplan p_{32} followed by action a_1 (i.e. $p_{31}; p_{32}; a_1$).
The intention selection mechanism with one step look ahead would be to enumerate each of the above plans p_1 and p_3 with respect to the set of constraints associated with each plan and objective function (i.e. *Maximize TC*) to determine the best intention to execute. A two-step look-ahead mechanism would look at maximizing the value of TC by enumerating plans p_{11} and p_{31} with the set of constraints associated with

1. plan p_1 along with subplan p_{11} on intention stack I_1;
2. plan p_3 along with subplan p_{31} on intention stack I_2.

Thus depending on n_i, the *number of steps* given by the programmer, the prioritization of intention is determined by the value of the objective function up to n_i levels in the intention stack for each intention. The one which yields the maximum value of the objective function would the the intention selected by S_I. In this case the tree generated is called the intention tree as shown in Figure 2 below where $I_1, I_2, \cdots I_n$ are the set of intention stacks.
The algorithm for selecting new intention using look-ahead is given below.

On the top of the selected intention there is a plan, and the formula in the beginning of its body is taken for execution. This implies that either a basic action is performed by the agent on its environment, an internal event is generated (in case the selected formula is an achievement goal denoted by $!g_i$), or a test goal is performed (which means that the set of beliefs has to be checked). If the intention is to perform a basic action or a test goal denoted by $?g_i$, the set of intentions needs to be updated. In the case of a test goal, the belief base will be searched for a belief atom that unifies with the predicate in the test goal. If that search succeeds, further variable instantiation will occur in the partially instantiated plan which contained that test goal (and the test goal itself is removed from the intention from which it was taken). In the case where a basic action is selected, the necessary updating of the set of intentions is simply to remove that action from the intention (the interpreter informs to the architecture component responsible for the agent effectors what action is required). When all formulae in the body of a plan have been removed (i.e., have been executed), the whole plan is removed from the intention, and so is the achievement goal that generated it (if that was the case).
This ends a cycle of execution, and CASO starts all over again, checking the state of the environment after agents have acted upon it, generating the relevant events, and so forth.

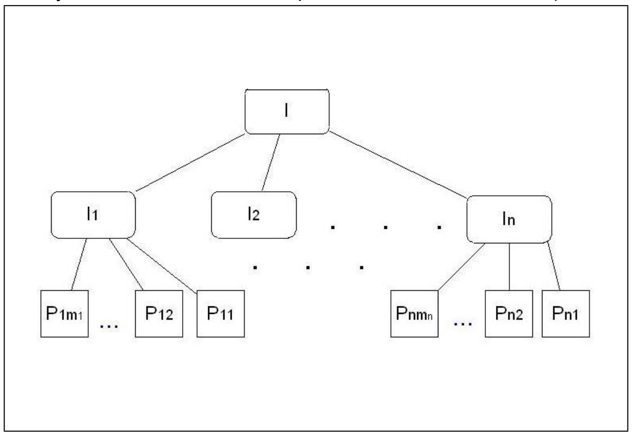


Figure 2: Intention Tree

Algorithm 2 LookAheadIntentionSelection(int n, ObjectiveStore OS, ConstraintStore CS))

```
1: Generate intention tree for all Intention Stacks.
2: Let constraint store at node $p = c_p$
3: Compute the value of objective function at the leaf nodes starting from left to right for each intention stack up to n nodes (i.e. n elements from top of Intention Stack) each by taking objectives and constraints from the objective store and the constraint store.
4: The best intention to execute is the intention stack which has the maximum value of the objective function at node n from the left.
```

5 Comparison and Conclusion

We now briefly summarize some of the work related to AgentSpeak(L) and BDI framework below.
Chalmers *et al.*(2001) constraint logic programming and data model approach is used within BDI agent framework. However, this work speaks of BDI agents in general and does not integrate with any BDI programming language. AgentSpeak(XL) programming language as described by Bordinin *et al.*(2002) integrates AgentSpeak (L) with the TAEMS scheduler in order to generate the intention selection function. It also describes a precise mechanism for allowing pro-

grammers to use events in order to handle plan failures which is not included in AgentSpeak(L). This work, however, adds priority to the tasks. Some related theoretical work on selecting new plans in the context of existing plans is presented by Horty *et al.*(2001). Another related work on detecting and resolving conflicts between plans in BDI agents is presented by Thangarajah *et al.*(2003). The degree of boldness of an agent, as defined by Schut *et al.*(2000), represents he maximum number of plan steps the agent executes before re-considering its intentions. However in this case it is assumed that the agent would backtrack if the environment changes after it has started executing the plans.
In this paper we have presented a general overview and informal discussion of the concept of incorporating constraints and objectives functions to AgentSpeak(L) as well as describe a means of how to design the option selection function for selecting a plan or an intention by using parametric look ahead mechanism. In future we would be extending CASO to incorporate inter-agent constraints in a multi-agent environment where agents may need to negotiate with each other.

References

Ooi, B. Hua & Ghose, A. K. (1999), Constraint-Based Agent Specification for a Multi-agent Stock Brokering System, *in* 'Multiple Approaches to Intelligent Systems:Proceedings of the 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems', Vol. 1611, Springer-Verlag Lecture Notes in Computer Science, pp. 409–419.

Jaffar, J. & Maher, M.J. (1994), Constraint logic programming: A survey., *in* 'Journal of Logic Programming' pp. 503–581

Kinny, D. & Georgeff, M. (1997), Modeling and design of multi-agent systems, *Intelligent Agents III, Lecture Notes in Artificial Intelligence Springer*, Berlin.

Morley., D. (1996), Semantics of BDI agents and their environment., *in* 'Tech. Rep. 74, Australian Artificial. Intelligent Institute, Melbourne'.

Rao, A.S. (1996), AgentSpeak(L): BDI agents speak out in a logical computable language, *in* 'Agents Breaking Away: Proceedings of the 7th European WS on Modelling Autonomous Agents in a Multi-Agent World', LNAI Vol 1038, pp. 42–55, Springer Verlag: Heidelberg, Germany.

Rao, A.S. & Georgeff, M. (1995), BDI Agents: from theory to practice., *in* 'Proceedings of First International Conference on Multi-Agent Systems', ICMAS-95, pp 312–319, San Francisco, CA.

Schut, M. & Wooldridge, M. (2000), Intention reconsideration in complex environments, *in* 'Proceedings of International Conference on Autonomous Agents', Barcelona, Spain.

Bordini, R.H., Bazzan, A.L.C., Jannone, R.O., Basso, D.M., Vicari, R.M. & Lesser, V.R. (2002), AgentSpeak(XL):Efficient intention selection in BDI agents via decisiontheoretic task scheduling, *in* Castelfranchi C. & Johnson, W.L. eds, 'Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems', ACM Press, pp 1294–1302, New York, USA.

Horty, J. & Pollack, M. (2001), Evaluating new options in the context of existing plans., *Artifical Intelligence*, Vol. 127, pp 199–220.

Machado, R. & Bordini, R.H. (2002), Running AgentSpeak(L) agents on SIMAGENT *in* Meyer J. & Tambe, M., eds.,'pre-proceedings of the 8th International Workshop on Agent Theories, Architectures and Languages', LNCS Vol. 2333, Springer Verlag: Berlin, Germany.

Lin, Z.-N., Hsu, H.-J. & Wang, F.-J. (2005), Intention Scheduling for BDI agents, *in* 'International Conference on Infomration Technology: Coding and Computing',ITCC-05, Vol-II.

Thangarajah, J., Padhgam, L. & Winikoff, M. (2003), Detecting and Avoiding Interference Between Goals in Intelligent Agents, *in* 'Proceedings of the 18th International Joint Conference on Artificial Intelligence',IJCAI 2003, pp. 721–726, Acapulco, Maxcio.

Thangarajah, J. (2004), Managing the Concurrent Execution of Goals in Intelligent Agents, Ph.D., RMIT, Australia.

Chalmers, S. & Gray, P.M.D. (2001), BDI agents and constraint logic, *AISB Journal Special Issue on Agent Technology* Vol. 1, pp. 21–40.

Dasgupta, A. & Ghose, A.K. (2005),Dealing with Objectives in a Constraint-Based Extension to AgentSpeak(L), *in* 'Eighth Pacific Rim Workshop on Multi-Agent Systems', PRIMA 2005.

Modelling Layer 2 and Layer 3 Device Bandwidths using B-Node Theory

S Cikara, S P Maj and DT Shaw
School of Computer and Information Science
Edith Cowan University
2 Bradford St Mount Lawley, Perth 6050, Western Australia
scikara@gmail.com; p.maj@ecu.edu.au; alidades@iinet.net.au

Abstract

Modern computer networks contain an amalgamation of devices and technologies, with the performance exhibited by each central to digital communications. Varieties of methods exist to measure and/or predict these performance characteristics. “Rule-of-Thumb” is subjective and based on prior experience, typically offering little mathematical rigour. Benchmarks use different scales and units, with comparative results possibly requiring further interpretation. Stochastic modelling uses complex mathematics which can be problematic and difficult to understand and conceptualise to the typical network administrator. As such, the specific technique employed depends on the problem domain and the cost of getting it wrong.

Bandwidth-Nodes (B-Nodes) are a high-level bandwidth-centric abstraction used to de-couple and control the complexity of a particular technology from the underlying implementation. Devices and/or technologies can be modelled as an individual node or as a collection of nodes, describing the overall function and interactions between both the sub-systems and the operating environment.

This paper uses a simple, common measurement method to calculate the theoretical maximum bandwidth of a single and/or collection of B-Nodes. It demonstrates that the efficiency of B-Nodes can be decomposed and shown as a product of all efficiencies contained within that node. Sub-optimal operation and device efficiency and its effect on bandwidth is also introduced. These are empirically validated and incorporated into the B-Node formula, allowing the bandwidth of a network to be calculated to a first approximation for a variety of devices and technologies. Hence, the anticipated performance of a network given a technical specification can be easily and quickly determined.

Keywords: modelling, B-Nodes, bandwidth, performance.

1 Introduction

A wide range of methods, terms, units and metrics are used to describe the performance of a network system. In conjunction with other factors such as price, they are used as an aid to selection (Maj and Veal 2001). In order to be of any practical value, they should be easy to understand and therefore be based on user perception of performance and as such, be simple and use reasonably sized units (Maj, Veal et al. 2000). Many of the results of these methods may require further interpretation and pose additional questions themselves. Others involve the use of complex mathematics and modelling, such as queuing theory, which can be problematic to analyse and difficult to understand and conceptualise to the typical network administrator.

Bandwidth-Nodes (B-Nodes) are a conceptually simple model used to control the detail of a system by the use of abstraction (Maj and Veal 2001). Details of the technical implementation are deliberately hidden as the specific technological execution may change rapidly and vary from device to device.

B-Nodes use a simple formula to determine the anticipated performance of individual components and networks as a whole. Recursive decomposition allows the performance of a node to be assessed by a simple, common measurement- bandwidth. Sub-optimal operation of B-Node efficiencies, including multiple compounded efficiencies, can also be introduced into an existing system, allowing the efficiency of a single or multiple B-Node(s) to be incorporated and evaluated right down to the device, protocol or technology level if so desired.

B-Node experimentation has shown the use of tools such as PING and File Transfer Protocol (FTP) to ascertain the bandwidth of a given configuration (Veal, Kohli et al. 2005). Work to date has not addressed the addition and subtraction of protocols and/or services to a specific device or configuration. This research will focus on empirically validating these variables and modelling each as its own individual sub-B-Node that impacts network performance, either positively or negatively.

Therefore, it is proposed that the anticipated performance of a network given a technical specification can be easily and quickly determined using B-Node modelling.

2 Network Performance

Network Performance is an amalgamation of terms, units and metrics used to characterise and quantify parameters such as delay, packet loss and bandwidth (Coccetti and Percacci 2002). As such, these cannot be simply expressed by a single parameter, and consequently there are numerous metrics and measurement methodologies employed to express such quantities.

As different applications place different requirements on a network, common criteria must be designed to maximise accurate common understanding by end users and service providers of the performance and reliability both of end-to-end paths and of specific 'IP clouds' (Paxson, Almes et al. 1998). For example, Voice over IP (VoIP) is an application that is sensitive to delay but requires relatively small bandwidth, and bulk data transfers that are insensitive to delay but require large bandwidth. As such, different metrics are used to measure the different quantities- delay is typically measured using packet loss and round trip time (Coccetti and Percacci 2002), (Padmanabhan, Qui et al. 2002), (Lai and Baker 2000), and bandwidth is typically measured by capacity, throughput and available bandwidth (Strauss, Katabi et al. 2003), (Lai and Baker 1999), (Prasad, Dovrolis et al. 2003), (Jain and Dovrolis 2002). Benchmarks can be used as an aid to answering these questions, however results may require further interpretation and additional questions may arise (Maj and Veal 2000).

Performance metrics must use concrete and well defined metrics, be repeatable, exhibit no bias for IP clouds using identical technology, exhibit fair and understood bias for IP clouds using non-identical technologies, avoid introducing artificial performance goals and be useful to users and providers in understanding the performance they experience or provide (Paxson, Almes et al. 1998).

Bandwidth, in a network-centric context, quantifies the data rate at which a network link or network path can transfer information (Prasad, Dovrolis et al. 2003). It must address the impact of application data plus overheads required to transport the data, all in a coherent and easily understood manner. Applications that depend on network capacity to transfer significant quantities of data over a single congestion-aware transport connection rely on the Bulk Transfer Capacity (BTC) of the network. BTC is defined as the long term average data rate over the path in question (Mathis and Allman 2001) and is hence defined as:

$$BTC = {}^{data\_sent}\!\big/_{elapsed\_time}$$

Therefore, the performance as perceived by the user, is constrained by the overall elapsed time an application takes to be executed over the underlying network (Mathis and Allman 2001).

BTC is an active measurement technique that directly probes network properties by generating the traffic required to make the measurement (Claffy and McCreary 1999). This active and direct method of analysis has the undesirable effect of the measurement traffic having a negative impact (saturation) on the performance of other traffic on the link (Coccetti and Percacci 2002), (Claffy and McCreary 1999).

As networks consist of heterogeneous devices and technologies (Maj and Kohli 2002) including computer nodes or hosts, network connection media, protocols, infrastructure and applications, interchanging any of these variables may vary network performance as each of these technologies have differing overheads. Subsequently, there exists a need for unbiased, empirical performance analysis that is simple, easy to use and conceptualise and be based on user perception of performance.

3 B-Nodes

Generally, any performance analysis or benchmark should provide a coherent conceptual model (Maj and Veal 2000). As such, the measurement standard used must be easy to understand, be based on user perception of performance, be simple, and utilize reasonably sized units (Maj, Veal et al. 2000).

Bandwidth Nodes, or B-Nodes, are a bandwidth-centric concept that uses high level abstraction to de-couple and hide the complexity of a particular technology from the underlying implementation (Maj and Veal 2001). They allow B-Nodes to be modelled as individual nodes (*Figure 1*) or as a sequence of nodes linked together (*Figure 2*).

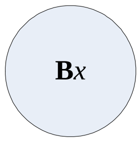


Figure 1 : B-Node

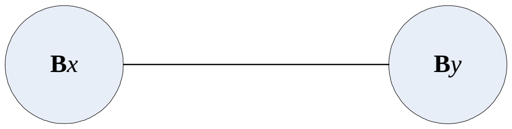


Figure 2: Interconnected B-Nodes

They also allow recursive decomposition to permit a device to be modelled as a collection of B-Nodes (*Figure 3*). A B-Node can also permit full or partial system or device overlap (*Figure 4*) (Maj, Veal et al. 2001).

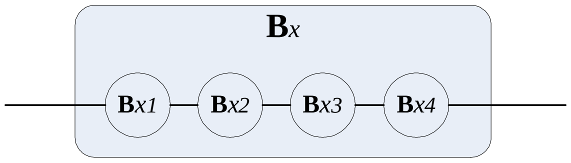


Figure 3: Recursive Decomposition

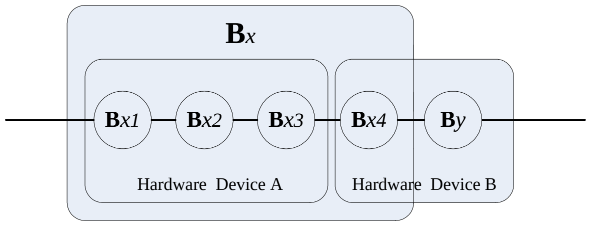


Figure 4: Partial device overlap

Furthermore, "*…each node … can now be now be treated as a quantifiable data source/sink … with associated transfer characteristics (Frames/s or Mbytes/s). This approach allows the performance of every node and data path to be assessed by a simple, common measurement-bandwidth. Where Bandwidth = Clock Speed x Data Path*

Width with the common units of Frames/s (Mbytes/s) … (Maj, Veal et al. 2000).

Operational constraints, including but not limited to processing capacity and interactions between slower nodes, typically influence B-Nodes to perform sub-optimally (Maj, Veal et al. 2001), (Maj and Veal 2001). As such, Maj et. al. has modified the simple bandwidth formula to incorporate sub-optimal operation using an "efficiency" multiplier. Therefore, the bandwidth of a B-Node is defined as:

Bandwidth = Clock Rate x Data Path Width x Efficiency or

$$B = C \times D \times E$$

Equation 1: B-Node formula showing sub-optimal operation

This formula can be applied to the theoretical maximum Bulk Transfer Capacity (and hence bandwidth) for TCP/UDP payloads over 100BASE-TX (100Mbps) Ethernet. All efficiency calculations within this paper are based on this reference protocol. Using the highest level of abstraction, 100BASE-TX has the following transmission characteristics:

Example 1: *Bandwidth* = ?

Clock Speed = 100 MHz

Data Path = $\frac{1}{8}B$ (converting bits into bytes), and

Efficiency = 1 (no transport overheads)

Hence:

$$B = 100MHz \times \frac{1}{8}B \times 1$$

$$B = 12.5MB/s$$

Using a lower level of abstraction, 100BASE-TX data encoding uses 4B/5B block coding which means that a 100Mb/s data stream requires 125Mb/s on the media (a 25% speedup resulting in 20% overhead or non-data bits transmitted) (Kaplan and Noseworthy 2000).

Example 2: *Clock Speed* =125 MHz (25% speedup)

Data Path = $\frac{1}{8}B$, and

Efficiency = $\frac{4}{5}$ (20% overhead for non-data bits transmitted)

And so:

$$B = 125MHz \times \frac{1}{8}B \times \frac{4}{5}$$

$$B = 12.5MB/s$$

Alternately, viewing the same problem from an even lower level of abstraction (after MLT-3 coding) the same formula now becomes:

Example 3: *Clock Speed* = $\frac{125}{4}MHz$

$= 31.25MHz$ (frequency is reduced to ¼) (Kaplan and Noseworthy 2000)

Data Path = $\frac{1}{8}B$, and

Efficiency= $\frac{4}{5} \times 4 = 3.2$ (20% overhead for non-data bits transmitted)

By reducing the carrier frequency without reducing the data rate, the efficiency is increased by a factor of 4. When it is demodulated at the other end, the efficiency is reduced by the same factor (4 times). So:

$$B = 31.25MHz \times \frac{1}{8}B \times 3.2$$

$$B = 12.5MB/s$$

From this we can see that the regardless of the level of abstraction, the formula still yields the same result- that being the maximum bandwidth of Ethernet is 12.5MB/s. For simplicity, all further calculations and assumptions are based on *Example 1*.

This high level abstraction only deals with 100BASE-TX and its effect on bandwidth. It does not address the subsequent reduction in efficiency additional network protocols and their associated overheads incur, in particular TCP/IP (hence referred to in this document as Ethernet).

4 The Internet Protocol (IP)

Internet Protocol version 4 (IPv4), developed in the 1980's (Information Sciences Institute 1981), is the most commonly used protocol in today's networks, and forms the integral basis for what we know as the Internet. As new and powerful applications using the Internet are developed, the underlying protocols operating in the lower layers of the OSI model (the networking protocol stack itself) remain unchanged (Xie 1999).

IPv4 is a network layer protocol that has provision for a 32-bit address space. Modern networks have surpassed IPv4's capabilities (Tanenbaum 1996). In order to address these and other shortcomings, Internet Protocol Version 6 (IPv6) has been developed (Deering and Hinden 1998) and is slowly being integrated into existing IPv4 infrastructure (Tanenbaum 1996).

IPv6 has a new simplified header format, including a 128-bit address space, which is designed to keep overhead to a minimum. The non-essential and optional fields have moved to extension headers that are placed after the IPv6 header. This reduces the common-case processing cost of packet handling and to limit the bandwidth cost of the new header (Deering and Hinden 1998) allowing for more efficient processing. IPv4 headers are not interoperable with IPv6 headers and hosts must implement both protocols in order to recognize and process both types of headers.

4.1 IP Overhead

By breaking down the various headers, we can analyse and predict the BTC performance degradation incurred between IPv4 and IPv6. By elaborating on Raicu and Zeadally's table (Raicu and Zeadally 2003), we can calculate the *Total Bytes of the Frame on the Wire* for an individual packet, as shown in *Table 1* (grey rows denote new fields introduced by the authors).

Packet Component	IPv4 TCP (B)	IPv6 TCP (B)	IPv4 UDP (B)	IPv6 UDP (B)
Preamble	7	7	7	7
Start of Frame Delimiter	1	1	1	1
Ethernet Header	14	14	14	14
IP Header	20	40	20	40
TCP/UDP Header	20	20	8	8
TCP/UDP Payload	1460	1440	1472	1452
Checksum	4	4	4	4
Interframe Gap	12	12	12	12
Total Overhead	78	98	66	86
Total Bytes of Frame on Wire	1538	1538	1538	1538
Efficiency (%)	94.93	93.63	95.71	94.41

Table 1: IPv4 and IPv6 header overhead showing both TCP and UDP

Using the *Total Bytes of Frame on Wire,* we can calculate the theoretical maximum single packet efficiency using the maximum data payload via *Equation 2*:

$$\text{SinglePacketEfficiency}(\%) = \frac{\text{TCP/UDP Payload}}{\text{Total Bytes of Frame on Wire}}$$

Equation 2: Theoretical maximum efficiency of a single packet

To evaluate the Bulk Transfer Capacity (bandwidth) of 100BASE-TX using these efficiency values, we get the results in *Table 2*:

	No Ethernet Protocol Overhead (Example)	IPv4 TCP	IPv6 TCP	IPv4 UDP	IPv6 UDP
Maximum Line Speed (Mb/s)	100	100	100	100	100
Maximum Line Speed (MB/s)	12.5	12.5	12.5	12.5	12.5
Efficiency of Ethernet (%)	100	94.93	93.63	95.71	94.41
Max Bulk Transfer Capacity (MB/s)	12.50	11.87	11.70	11.96	11.80

Table 2: Bulk Transfer Capacity of IPv4 and IPv6

Using the efficiency percentages from *Table 1*, we obtain the efficiency of a specific protocol (E_{Ethernet}) and from this, the computed theoretical maximum BTC for a single protocol (or B-Node) is calculated for 100BASE-TX (*Table 2*). However, the simple B-Node formula (*Equation 1*) does not address multiple B-node efficiencies. It must be extrapolated further to combine the effects of multiple efficiencies and their influence on node bandwidth.

5 B-Node Efficiency Decomposition

By further decomposing *Equation 1*, the efficiency of the B-Node (E) can be shown as a product of all efficiencies (e_i) contained within the B-Node (*Equation 3*).

$$E = \prod_{i=1}^{n} e_i$$

Equation 3: B-Node efficiency product formula

For each B-Node, there is the absolute efficiency, which is the ratio of input to output of each individual B-node, and a relative efficiency which compares the reference value to the output of the B-Node. An example is shown in *Figure 5*.

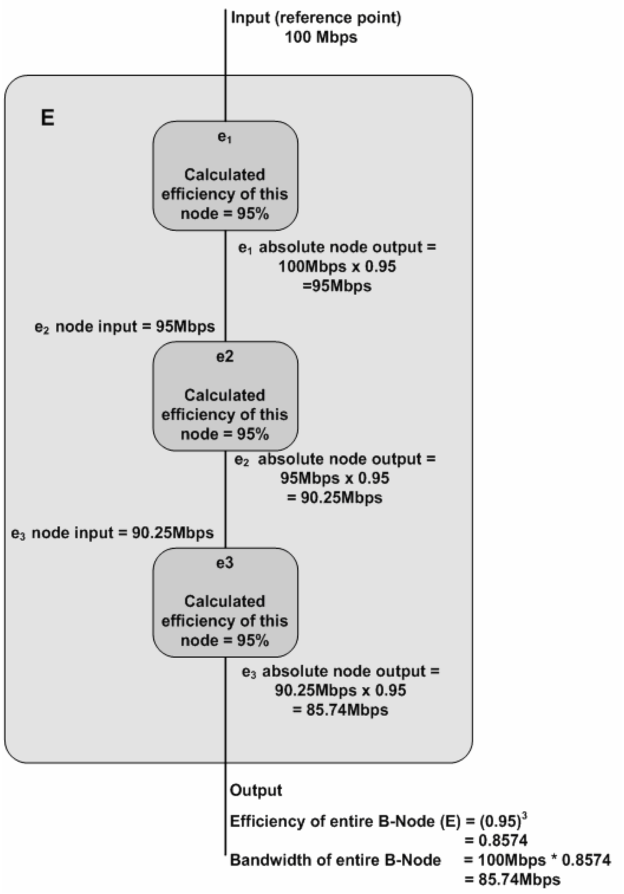

Input (reference point)
100 Mbps
E
e1
Calculated efficiency of this node = 95%
e1 absolute node output = 100Mbps x 0.95 =95Mbps
e2 node input = 95Mbps
e2
Calculated efficiency of this node = 95%
e2 absolute node output = 95Mbps x 0.95 = 90.25Mbps
e3 node input = 90.25Mbps
e3
Calculated efficiency of this node = 95%
e3 absolute node output = 90.25Mbps x 0.95 = 85.74Mbps
Output
Efficiency of entire B-Node (E) = (0.95)^3 = 0.8574
Bandwidth of entire B-Node = 100Mbps * 0.8574 = 85.74Mbps

Figure 5: B-Node decomposition example

The component efficiencies (e_i) can further be divided dependent on whether the additional overhead contains Control Packet information or Data Packet overheads.

Data Packet overheads are defined as overheads that are directly added to packets that are transmitting application data. One such example includes Virtual Local Area Network (VLAN) tags. Therefore, e_i, to a first approximation now becomes:

$$E = e_i \times (1 - e_i \times \Delta e_{i+n})$$

$$\text{where} \quad \Delta e_{i+n} = \frac{\text{Additional Data Packet Overhead}}{\text{Data payload}}$$

Equation 4: Data packet efficiency equation

Control Packet information is defined as entirely additional packets used to control link flow. They carry no user data and can typically be viewed as packets that reduce the bandwidth of a link, without transmitting any real application data. Some examples include Spanning-Tree Protocol (STP), Routing Information Protocol (RIP), Enhanced Interior Gateway Routing Protocol (EIGRP) and Open Shortest Path First (OSPF). In this situation, e_{i+n} to a first approximation becomes:

$$e_{i+n} = (1 - \alpha e_{i+n})$$

$$\text{where} \quad \alpha e_{i+n} = \frac{\text{Control packet size per second}}{\text{Link speed per second}}$$

Equation 5: Control packet efficiency equation

The original B-Node formula remains the same, however the e_{i+n} parameter can be interchanged with as many Control Packet or Data Packet efficiencies as required to be added.

To remove an efficiency from a already calculated B-Node, this can simply be achieved by multiplying the B-Node efficiency with the inverse of the efficiency to be removed (*Equation 6*):

$$\frac{1}{e_{i+n}}$$

Equation 6: Efficiency removal equation

This can be applied to both Control and Data Packet efficiencies.

For example, using a B-Node with 5 sub-nodes as defined below:

e_1 is Ethernet Efficiency

e_2 and e_5 are Data Packet efficiencies

e_3 and e_4 are Control Packet Efficiencies

The B-Node formula (*Equation 1*) now becomes:

$$B = C \times D \times (e_1 \times e_2 \times e_3 \times e_4 \times e_5)$$

$$B = C \times D \times (e_1 \times (e_1 \times (1 - (e_1 \times \Delta e_2))) \times (1 - \alpha e_3) \times (1 - \alpha e_4) \times (1 - (e_1 \times \Delta e_2)))$$

As all devices are not created equally, each with their own technological constraints, the B-Node formula does not cater for individual device efficiencies. As such, it must be further expanded to account for these variations in device implementations.

5.1 Device Sub-Optimal operation and its effect on Bandwidth

In an ideal system, an intermediary device such as a switch, router or bridge would have little or no impact on bandwidth. However, this is not always the case. A device itself can introduce latency or processing overheads within a link and hence reduce bandwidth. This may be particularly pronounced in computationally intensive operations such data encryption and decryption.

It is envisaged that there is no one single figure (e_{Di}) for an entire device, rather a figure for each process the device purports to undertake. For example, a router might be particularly fast at switching IPv4 packets, but not very fast at IPv4 encryption using Advanced Encryption Standard (AES) with 256 bit keys. As such, these must be addressed individually. The Efficiency parameter now becomes:

$$E = \prod_{i=1}^{n} e_i e_{Di}$$

Equation 7: B-Node efficiency formula with device sub-optimal operation

The B-Node formula is extrapolated again to take into account this device sub-optimal operation:

$$B = C \times D \times (e_1 e_{D1} \times e_2 e_{D2} \times \ldots\ldots \times e_{i+n} e_{Di+n})$$

Equation 8: Extrapolated B-Node equation

Using empirically derived results, e_{Di} for an individual process on a particular device can be evaluated.

6 Empirical Validation

6.1 Initial Benchmarking

The initial test bed consisted of two identical 800MHz Celeron dual-stack IBM-compatible PCs with Windows 2003 Enterprise operating system installed. The Intel Pro 100S network interface cards of each machine were directly connected to each other via a crossover cable. This setup (*Figure 6*) forms the benchmark baseline.

To empirically measure the Bulk Transfer Capacity of a link (and hence evaluate B-Nodes), there was a requirement for a single program that could perform IPv4, IPv6, TCP and UDP measurements. In addition to this, it was identified that the performance of a BTC program is often limited by the speed of a disk drive (Spurgeon 2000). Furthermore, the program had to account for this by performing memory-to-memory data transfers. Iperf (NLANR Distributed Application Support Team 2003) was initially evaluated, however erroneous results for IPv6 UDP transfers rendered the program inadequate for the purposes of this experimentation. As such, nuttcp (Fink and Scott 2004) was assessed to meet all the aforementioned requirements. The program's documentation describes "*…its most basic usage is to determine the raw TCP (or UDP) network layer throughput by transferring memory buffers from a source* *system across an interconnecting network to a destination system, either transferring data for a specified time interval, or alternatively transferring a specified number of buffers."*

6.1.1 Initial Benchmark Results (PC to PC)

Using the above method, the efficiency of any introduced or removed system can be calculated and validated. In this case, the efficiency of a BTC test between two identical PCs connected via a cross-over cable was to be assessed. This relative measurement for a minimalistic system was important as it demonstrated the maximum transfer characteristics of an "unloaded" node. All other measurements are calculated relative to these values, either directly or indirectly.

Figure 6 shows the experimental setup consisting of three B-Nodes. The centre node consists of the two identical PCs, the second most inner node is made of Ethernet efficiency (which has already been calculated in *Table 2*), and the outer B-Node, which is the overall efficiency of the node in relation to the input (reference point) and the output (measuring point). This measured value, in conjunction with Ethernet efficiency, allows the empirical calculation of the inner node, and the node efficiency for that specific hardware setup.

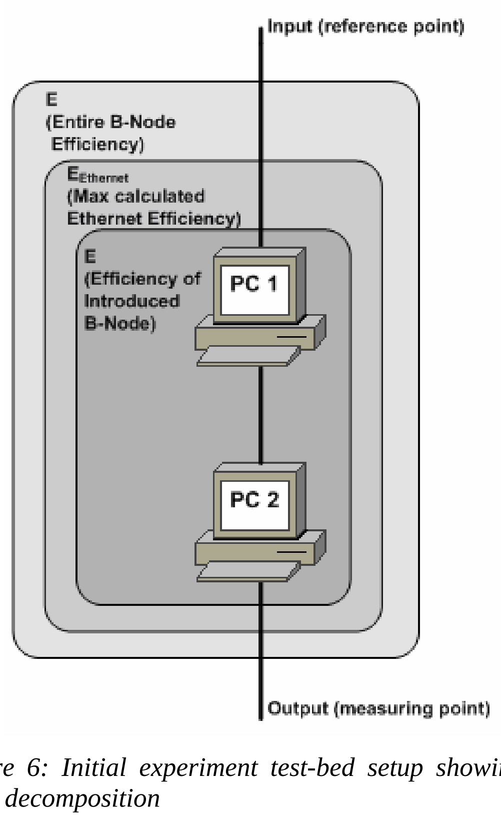

Input (reference point)
E
(Entire B-Node Efficiency)
E_Ethernet
(Max calculated Ethernet Efficiency)
E
(Efficiency of Introduced B-Node)
PC 1
PC 2
Output (measuring point)

Figure 6: Initial experiment test-bed setup showing B-Node decomposition

From the results (*Table 3*), it can be concluded that both IPv4 and IPv6 TCP have almost optimal (or 100%) efficiencies compared to the calculated value, with both being above 99.78%. Both UDP transfers perform slightly worse than their TCP counterparts (at best almost 1% less) with IPv6 UDP (98.48%) approximately a further 0.5% less than IPv4 UDP (98.93%).

	Theoretical Maximum (MB/s) (from Table 2)	Ouput at measuring point (MB/s)	Difference (MB/s)	Max calculated Ethernet Efficiency (%)	Actual entire B-Node Efficiency (%)	Actual Efficiency (Ed) of Introduced B-Node (%)
IPv4 TCP	11.87	11.84	0.03	94.93	94.75	99.78
IPv4 UDP	11.96	11.83	0.13	95.71	94.66	98.93
IPv6 TCP	11.70	11.68	0.02	93.63	93.43	99.82
IPv6 UDP	11.80	11.62	0.18	94.71	92.96	98.48

Table 3: Bulk Transfer Capacity of IPv4 and IPv6 showing actual efficiency of introduced B-Node.

6.2 Layer 2 Device Measurement

6.2.1 Single Switch Experiments

To calculate specific device efficiencies, the experiment was further elaborated to incorporate both unmanaged (DLink DES1008D) and managed (Cisco 2950 and 3550 series) switches. The equipment was set up as shown in *Figure 7*.

As managed switches have more features available than unmanaged switches, the opportunity to individually test the efficiencies of these was investigated. Initially, a Cisco default switch configuration was tested. In this case, the PCs were in Virtual Local Area Network 1 (VLAN 1), and Spanning-Tree Protocol (STP) was enabled. Various combinations of these were then evaluated including:

1. PCs in VLAN 1 and STP disabled
2. PCs in VLAN 1 and STP enabled (Cisco default configuration)
3. PCs in VLAN 10 and STP disabled
4. PCs in VLAN 10 and STP enabled

Note: On a switch, access ports or non-trunking ports have no VLAN information passed on them. The VLAN tags are not passed through to the PC (and hence do not occupy any time on the wire) and as such, should not impact bandwidth.

It should also be noted that the experiments were conducted using a stable and settled STP network with hello timers set to the default of 2 seconds. Using these parameters, we obtain a calculated maximum efficiency for an STP B-Node to be 99.999663%. This should have negligible impact on BTC.

The experimental setup (*Figure 7*) in this instance consists of four B-Nodes, but with a variable number of ei_{+n} sub-nodes shown in the switch. These variable numbers of sub-nodes in the switch pertain to device specific functionality, such as VLANs and STP. Building up on the methodology introduced in Section 6.1.1, the measured output allows the empirical derivation of the e_{i+n} sub-nodes, and hence, the specific node efficiency for a particular hardware setup, as well as a particular protocol or configuration activated and operating on that device.

From the results obtained, we can see that the introduced B-Node efficiency for both managed and unmanaged switches, regardless of protocols used, was overall fairly constant and close to optimal efficiency (with the minimum being 99.58%, the average being 99.89%). There were instances where the efficiency was greater than 100% (the maximum being 100.6%), but to a first approximation, these can be accounted for in measurement, rounding errors and uncertainties. As such, it can be determined that the addition of a switch within a B-node will have negligible or no effect on Bulk Transfer Capacity.

Analysing the switch sub-node efficiencies also demonstrated that regardless of the VLAN or if STP was enabled or disabled, the result was a negligible impact on bandwidth. The sub-node efficiencies ranged from 99.66% to 100.17%, with an average of 99.97%.

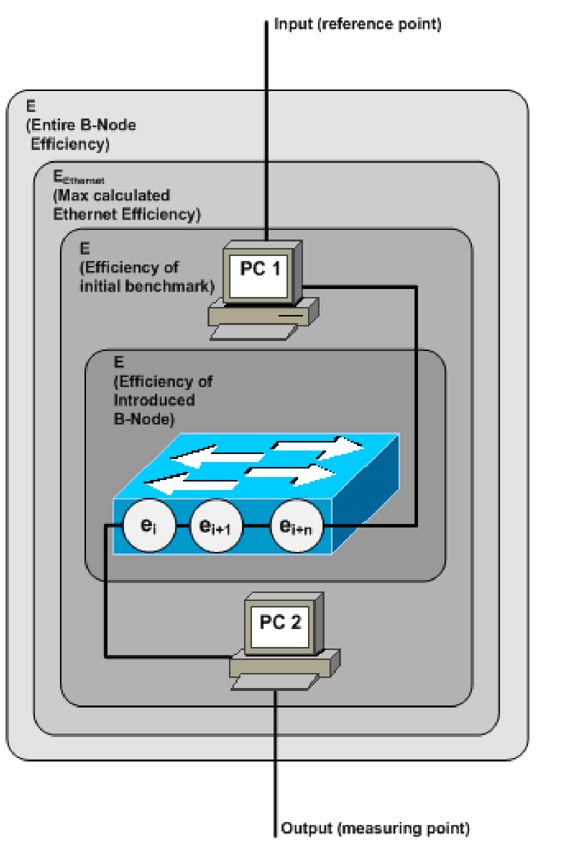

Input (reference point)
E (Entire B-Node Efficiency)
$E_{Ethernet}$ (Max calculated Ethernet Efficiency)
E (Efficiency of initial benchmark)
PC 1
E (Efficiency of Introduced B-Node)
e_i
e_{i+1}
e_{i+n}
PC 2
Output (measuring point)

Figure 7: Single switch experiment setup

6.2.2 Dual Switch Experiments

The experiment was then further extended to incorporate two unmanaged (DLink DES1008D) or two managed (Cisco 2950 and 3550 series) switches. The equipment was set up as shown in *Figure 8*.

The additional features that were tested are listed below:

1. PCs in VLAN 1, using 802.1Q encapsulation and STP disabled
2. PCs in VLAN 1, using 802.1Q encapsulation and STP enabled (default configuration)
3. PCs in VLAN 10, using 802.1Q encapsulation and STP disabled
4. PCs in VLAN 10, using 802.1Q encapsulation and STP enabled
5. Same combinations as above, but using Inter-Switch Link (ISL) for encapsulation

The experimental setup (*Figure 8*) again shows four B-Nodes, and a variable number of e_{i+n} sub-nodes. The variable numbers of sub-nodes are setup-specific functionality, such as VLANs and STP and encapsulation type. The specific node efficiency for a particular hardware setup as well as a particular protocol or configuration activated and operating on that device was then evaluated.

From the results, excluding IPv6 TCP on the 3550 (reasons explained further on), we can see that introduced B-Node efficiency is overall fairly constant for both dual managed and unmanaged switches for the features tested (average of 99.92%).

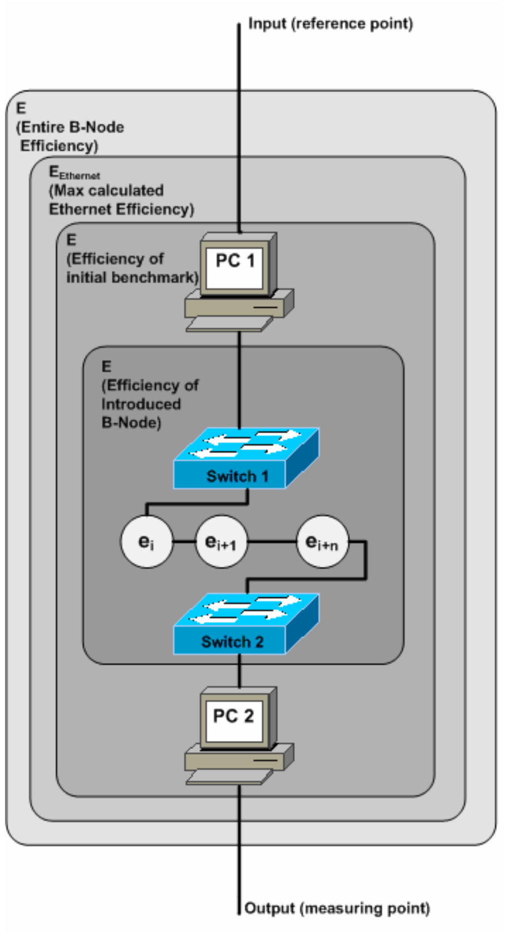

Input (reference point)
E (Entire B-Node Efficiency)
$E_{Ethernet}$ (Max calculated Ethernet Efficiency)
E (Efficiency of initial benchmark)
PC 1
E (Efficiency of Introduced B-Node)
Switch 1
e_i
e_{i+1}
e_{i+n}
Switch 2
PC 2
Output (measuring point)

Figure 8: Dual switch experiment setup

On the Cisco 3550 using IPv4 TCP with ISL encapsulation, the efficiency also varies the greatest with respect to the reference value (96.84% and 98.99%). IPv4 and IPv6 UDP with VLAN tagging and ISL encapsulation also had an efficiency that is greater than what can be accounted for in measurement, rounding errors and uncertainties (101.76% to 101.86%). This indicates that the use of these protocols *increases* the efficiency of the B-Node. Possible explanations for this may include the Cisco implementation of these protocols. Further research is required to investigate this phenomena.

IPv6 TCP bandwidth for the Cisco 3550 was significantly lower than for the Cisco 2950 switch (average of 81.62% with approximate 1% deviation from minimum to maximum). Further research is required to explain this, however one possible solution is the software implementation of this particular Internetworking Operating System (IOS) of this switch and its interaction with the congestion control algorithms of IPv6 TCP. This demonstrates that the device efficiency (e_{Di}) for a Cisco 3550 switch using IPv6 TCP is *significantly lower* than for any of the other devices tested.

The sub-node efficiencies (e_{i+n}) showed also that regardless of VLAN, encapsulation or if STP was enabled or disabled, the result was an insignificant effect on bandwidth. The sub-node efficiencies ranged from 99.16% to 101.95%, with an average of 100.22%.

It can be seen from these results that the device (e_{Di}) in conjunction with the protocol used ($e_{Ethernet}$) has the greatest effect on Bulk Transfer Capacity. Ancillary protocols or features (such as STP, encapsulation type and VLANs) have little or no effect on bandwidth. This information would be particularly valuable to a network administrator evaluating and planning network infrastructure.

6.3 Layer 3 Devices

6.3.1 Single Router Experiments

The effect of Layer 3 devices on bandwidth was next to be investigated and empirically evaluated. The device assessed in these experiments was a 2621XM Cisco router, and setup as in *Figure 7* (but with the switch replaced with the router). The results are shown in *Table 4*.

	Theoretical Maximum (MB/s)	Output at measuring point (MB/s)	Difference (MB/s)	Max Ethernet Efficiency (%)	Actual Entire B-Node Efficiency (%)	Efficiency of Introduced B-Node (%)	Efficiency of Introduced sub-B-Node (ei+n)
IPv4 TCP NoCEF	11.87	7.91	3.96	94.93	63.28	66.79	Ref. Val
IPv4 TCP CEF	11.87	7.92	3.95	94.93	63.36	66.87	100.13
IPv4 UDP NoCEF	11.96	7.01	4.95	95.71	56.08	59.24	Ref. Val
IPv4 UDP CEF	11.96	7.04	4.92	95.71	56.32	59.50	100.43
IPv6 TCP NoCEF	11.70	2.5	9.20	93.63	20.00	21.41	Ref. Val
IPv6 TCP CEF	11.70	7.39	4.31	93.63	59.12	63.28	295.60
IPv6 UDP NoCEF	11.80	1.27	10.53	94.71	10.16	10.93	Ref. Val
IPv6 UDP CEF	11.80	7.07	4.73	94.71	56.56	60.84	556.69

Table 4: Single Router Bandwidths

The effect the router has on bandwidth is much more pronounced than a switch. With the exception of IPv6 without using Cisco Express Forwarding (CEF), average TCP bandwidth (65.65%) is consistently higher than UDP (59.86%), with IPv4 TCP (average of 66.83%) having a greater efficiency than IPv6 TCP (63.28%) by approximately 3.5%. Conversely to this, IPv4 UDP (59.37%) is lower than IPv6 UDP (60.84%) by about 1.5%.

Disabling CEF and using IPv6 has the greatest effect on overall router bandwidth. With IPv6 TCP, the efficiency was reduced to 21.41%. IPv6 UDP was approximately 51% less efficient than IPv6 TCP with 10.93%

More pronounced is the effect sub-nodes have on IPv6 router efficiency. By enabling CEF on IPv6 TCP, the efficiency is almost trebled to 295.60%. The result of enabling CEF on IPv6 UDP Bulk Transfer Capacity is even more significant, at 556.69%. Less distinctive is the effect of CEF on IPv4, with the sub-node contributing less than a 0.5% increase in efficiency.

6.3.2 Dual Router Experiments

To quantify the effect of multiple layer 3 devices, 2621XM Cisco routers were paired up and the results noted as follows. In addition, a single Access Control List statement (ACL) was applied to the in and out direction of the ingress interface of *Router 1* and egress interface of *Router 2*. Experimental setup was as in *Figure 8*, with the switches replaced with routers. *Table 5* displays the results.

Average IPv4 TCP performance using dual routers (67.12%) compared favourably with single routers (66.84%), as did IPv4 UDP (dual routers 60.19%) and single routers (59.37%).

Excluding the results obtained from using no CEF, and ACL statements, IPv6 TCP dual routers (DR) were approximately lower by 10.5% than with single routers (SR), to 52.74%.

IPv6 TCP with no CEF was also fairly comparable (DR 20.12% compared to SR 21.41%), as was IPv6 UDP with no CEF (DR 11.10% to SR 10.93%).

Single ACL statements also have significant impact on IPv6 efficiencies. For IPv6 TCP, the statement reduces bandwidth by 13.27% to 39.47%. With IPv6 UDP, this was only reduced by 8% to 52.41%. IPv4 ACL statements improved efficiency by less than 0.6%, which can be accounted for in errors and rounding.

The sub-node efficiencies (e_{i+n}) for IPv4 demonstrated that CEF or an ACL statement does not have an appreciable effect on bandwidth. IPv6 TCP showed that with the introduction of two routers with CEF enabled, efficiency increased to 262.13%, but the addition of an ACL statement reduced this by almost 66% to 197.17%. IPv6 UDP with CEF enabled increased to 544.19% and an ACL statement reduced this by 72.1% to 472.09%

From the results it can be seen that the device sub-nodes (e_{i+n}) in conjunction with the protocol used ($e_{Ethernet}$) has a significant effect on Bulk Transfer Capacity in routers.

	Theoretical Maximum (MB/s)	Measured (MB/s)	Difference (MB/s)	Max Ethernet Efficiency (%)	Actual Entire B-Node Efficiency (%)	Efficiency of Introduced B-Node (%)	Efficiency of Introduced sub-B-Node (ei+n)
IPv4 TCP NoCEF	11.87	7.93	3.94	94.93	63.44	66.95	Ref. Val
IPv4 TCP CEF	11.87	7.92	3.95	94.93	63.36	66.87	99.87
IPv4 TCP CEF ACL	11.87	8.00	3.87	94.93	64.00	67.55	100.88
IPv4 UDP NoCEF	11.96	7.05	4.91	95.71	56.40	59.94	Ref. Val
IPv4 UDP CEF	11.96	7.07	4.89	95.71	56.56	60.11	100.28
IPv4 UDP CEF ACL	11.96	7.12	4.84	95.71	56.96	60.53	100.99
IPv6 TCP NoCEF	11.70	2.35	9.35	93.63	18.80	20.12	Ref. Val
IPv6 TCP CEF	11.70	6.16	5.54	93.63	49.28	52.74	262.13
IPv6 TCP CEF ACL	11.70	4.61	7.09	93.63	36.88	39.47	196.17
IPv6 UDP NoCEF	11.80	1.29	10.51	94.71	10.32	11.10	Ref. Val
IPv6 UDP CEF	11.80	7.02	4.78	94.71	56.16	60.41	544.19
IPv6 UDP CEF ACL	11.80	6.09	5.71	94.71	48.72	52.41	472.09

Table 5: Dual Router Bandwidths

6.4 B-Node Network Performance Analysis

A fictitious network administrator has been given the task to analyse the network shown in *Figure 9*, and to use B-Node methodology to predict the performance of the topology. The technical specification is detailed as below:

1. *PC 1*, *2* and *3* are all identical 800MHz PCs
2. *Switch 1* is a DLINK DES1008D switch
3. *Switch 2* and *3* are Cisco 3550 switches
4. *Router 1*, *2* and *3* are Cisco 2621XM routers

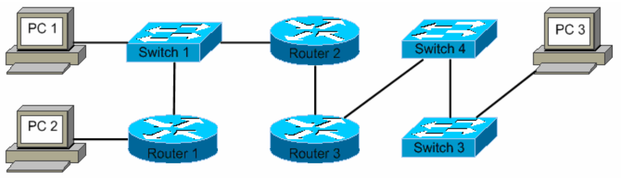


Figure 9: Fictitious network

Assuming no competing transfers, the administrator wants to evaluate the anticipated performance between PC 1 and PC 2 using IPv6 UDP. Router 1 does not use CEF. The B-Nodes for this configuration are:

1. IPv6 UDP Ethernet
2. PC to PC
3. DLink DES 1008D switch, and
4. Cisco 2621XM with no CEF (and hence no sub-nodes)

The B-Node formula hence becomes:

$$B = 100 \times \tfrac{1}{8} \times \begin{pmatrix} e_{IPv6UDP\,Ethernet} \times e_{PC\,to\,PC} \times \\ e_{DLink} \times e_{(Router1\,No\,CEF)} \end{pmatrix}$$

Using the empirically derived results from this research, we get:

$$B = 100 \times \tfrac{1}{8} \times (0.9441 \times 0.9848 \times 100.60 \times 10.16)$$

$$B = 1.19 MB/s$$

The anticipated bandwidth of this configuration is 1.19MB/s. The experimental result obtained was 1.23MB/s which compares favourably with the predicted result.

The network administrator now wants to evaluate the bandwidth between PC 1 and PC 3 using IPv4 TCP transfers, again assuming no competing transfers. Switch 2 and 3 use VLANs, ISL encapsulation but no STP. Router 2 and 3 use CEF. The B-Nodes now are:

1. IPv4 TCP Ethernet
2. PC to PC
3. DLink DES 1008D switch
4. Router to Router (Cisco 2621XM) with a CEF sub-node, and
5. Dual 3550 switches and ISL encapsulation with the sub-node VLANs

The B-Node formula becomes:

$$B = 100 \times \tfrac{1}{8} \times \begin{pmatrix} e_{IPv4TCP\,Ethernet} \times e_{PC\,to\,PC} \times e_{DLink} \times \\ \left(e_{(Router\,to\,Router)} e_{(Router\,to\,Router\,CEF)}\right) \times \\ \left(e_{(Dual\,3550 + ISL\,encapsulation)} e_{(Dual\,3550 + VLANs)}\right) \end{pmatrix}$$

Using the empirically derived results from this research, we get:

$$B = 100 \times \tfrac{1}{8} \times (0.9493 \times 0.9978 \times 0.9988 \times (0.6695 \times 0.9987) \times (0.9701 \times 101.95))$$

$$B = 7.82 MB/s$$

The anticipated calculated bandwidth of this configuration between PC1 and PC3 is 7.82MB/s. Results obtained experimentally compared well to the calculated figure to a first approximation, with a 7.92MB/s bandwidth obtained.

From this, we can see that for a given technical network specification and using B-node analysis, the expected bandwidth for non competing transfers can be calculated to a first approximation.

6.5 Conclusion

B-Nodes may provide a simple, easy to use diagrammatic tool that can be used to hide the complexity of devices and technologies and the performance exhibited by them. Through the use of abstraction, the complexity of a particular technology and its implementation can be decoupled and controlled, allowing them to be modeled as an individual node, or as a collection of nodes showing the overall system structure.

Using the B-Node methodology and its empirical validation, specific technology and device efficiencies have been evaluated and calculated. By decomposing the elements within a configuration and using this information, the simple B-Node formula allows the bandwidth of a network to be calculated to a first approximation, down to the individual components if so desired. Each network communication device, computer nodes or hosts, network connection media and protocols, may be evaluated as required and using this information, the anticipated network performance, given a technical specification, can be easily and quickly determined using the simple B-Node formula, however further investigation and empirical validation of a wider variety of protocols and hardware platforms is required.

7 References

Claffy, K. C. and S. McCreary: (1999): Internet measurement and data analysis: passive and active measurement, http://www.caida.org/outreach/papers/1999/Nae4hansen/Nae4hansen.html

Coccetti, F. and R. Percacci (2002). Bandwidth Measurement and Router Queues. Trieste, Sezione de Trieste.

Deering, S. and R. Hinden: (1998): RFC 2460 Internet Protocol Version 6 (IPv6) Specification, http://www.rfc-editor.org

Fink, B. and R. Scott: nuttcp, v5.1.11 ftp://ftp/lcp.nrl.navy.mil/pub/nuttcp/ 2004

Information Sciences Institute: (1981): RFC 791 Internet Protocol, http://www.rfc-editor.org

Jain, M. and C. Dovrolis (2002). End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. SIGCOMM, Pittsburgh, Pennsylvania, USA.

Kaplan, H. and B. Noseworthy: (2000): The Ethernets: Evolution from 10 to 10,000 Mbps- How it all Works!, http://www.iol.unh.edu/training/ethernet.html

Lai, K. and M. Baker (1999). Measuring Bandwidth. 18th Annual Joint Conference of the IEEE Computer and Communications Societies.

Lai, K. and M. Baker (2000). Measuring Link Bandwidth Using a Deterministic Model of Packet Delay. Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden, ACM Press.

Maj, S. P. and G. Kohli (2002). Modelling Global IT Structures using B-Nodes. 3rd Annual GITM World Conference, New York, USA.

Maj, S. P. and D. Veal (2000). Architecture Abstraction as an Aid to Computer Technology Education. ASEE Computers in Education Division, St Louis, Missouri, USA.

Maj, S. P. and D. Veal (2001). B-Nodes: A proposed new method for modelling information system technology. International Conference on Computing and Information Technologies, Montclair State University, NJ, USA.

Maj, S. P. and D. Veal (2001). Controlling Complexity in Information Technology: Systems and Solutions. IASTED Conference on Computers and Advanced Technology in Education (CATE), Banff, Canada.

Maj, S. P., D. Veal and P. Charlesworth (2000). Is Computer Technology Taught Upside Down? 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland.

Maj, S. P., D. Veal and R. Duley (2001). A Proposed New High Level Abstraction for Computer Technology. ACM Special Interest Group for Computing Science Education (SIGCSE) 2nd Technical Symposium in Computer Science Education, Charlotte, North Carolina, USA.

Mathis, M. and M. Allman: (2001): RFC: 3148 A Framework for Defining Empirical Bulk Transfer Capacity Metrics, www.rfc-editor.org

NLANR Distributed Application Support Team: Iperf, http://dast.nlanr.net 2003

Padmanabhan, V. N., L. Qui and H. J. Wang (2002). Technical Report MSR-TR-2002-39: Server Based Inference of Internet Performance, Microsoft Research, Microsoft Corporation: Redmond, WA.

Paxson, V., G. Almes, J. Mahdavi and M. Mathis: (1998): RFC 2330 Framework for IP Performance Metrics, www.rfc-editor.org

Prasad, R., C. Dovrolis, M. Murray and K. C. Claffy (2003). Bandwidth Estimation: Metrics, Measurement Techniques, and Tools. IEEE Network. **17:** 27- 35.

Raicu, I. and S. Zeadally (2003). Impact of IPv6 on End-User Applications. IEEE International Conference on Telecommunications (ICT), Tahiti, French Polynesia.

Spurgeon, C. E. (2000). Ethernet: The Definitive Guide, Library of Congress Cataloguing-in-Publication Data.

Strauss, J., D. Katabi and F. Kaashoek (2003). A Measurement Study of Available Bandwidth Estimation Tools. Proc. ACM SIGCOMM Conference on Internet Measurement, Miami Beach, FL.

Tanenbaum, A. S. (1996). Computer Networks. Upper Saddle River, N.J, Prentice Hall.

Veal, D., G. Kohli, S. P. Maj and J. Cooper (2005). A Framework for a Bandwidth Based Network Performance Model for CS Students. 2005 ASEE Annual Conference and Exposition "The Changing Landscape of Engineering and Technology Education in a Global World", Portland, Oregon.

Xie, P. P. (1999). Network Protocol Performance Evaluation of IPv6 for Windows NT. San Luis Obispo, California Polytechnic State University.

Throughput fairness in k-ary n-cube networks

Cruz Izu
School of Computer Science
The University of Adelaide
Adelaide 5001, South Australia
cruz@cs.adelaide.edu.au

Abstract

The performance of an interconnection network is measured by two metrics: average latency and peak network throughput. Network throughput is the total number of packets delivered per unit of time.
Most synthetic network loads consist of sources injecting at the same given rate, using traffic patterns such as random, permutations or hot spot, which reflect the distribution of packet destinations in many parallel applications. The network is assumed to be fair: all source nodes are able to inject at the same rate. This work will show such assumption is unfounded for most router proposals. All router designs exhibited significant network unfairness under non-uniform loads. Some routers are also unfair under random traffic patterns. At loads above saturation, if the channel utilization is uneven, the injection matrix will become uneven: packet at low used areas will be accepted at a higher rate that those at the busy areas.
As synthetic traffic does not reflect the coupled nature of the traffic generated by parallel applications, the impact of this unfairness on application performance could not be measured. New synthetic loads need to be developed to better evaluate network response beyond saturation.

Keywords: Interconnection Networks, network throughput, fairness, channel utilization.

1 Introduction

Massively Parallel Processors (MPPs) are built by connecting a large number of common microprocessors with off-the shelf interconnect technologies such as Myrinet or Quadrics (Pretini et al, 2002) or custom designed network such as those built into the BlueGene (Blomrich et al 2003) or the Cray XT3. In addition of managing message traffic for parallel applications, the interconnection network (IN) provides support for data distribution, periodic check-pointing, input/output handling and results storage.

The first step to design an interconnection network is to choose the network topology. Low degree networks are popular as they map easily into the plane, which facilitates implementation. Besides, if we are bandwidth limited, lower node degree results in wider physical channels that reduce message transmission time. K-ary n-cube networks of degree 2 or 3 are a popular choice (Duato, Yalamanchili and Ni, 1997) as they provide a regular symmetric direct network with the advantages this entails.

The IN should provide low latency and high throughput, and avoid any network anomalies such as deadlock, livelock or starvation. There is a large body of result in dealing with deadlock issues either by deadlock avoidance or deadlock recovery. Most of the solutions are based on restricting routing to eliminate cyclic dependencies (Duato, Yalamachili and Ni, 1997) or to break the possible cycles by mapping messages to separate virtual channels (Dally and Seitz, 1987). Once we achieve a deadlock-free network design it is easy to increase adaptivity by applying Duato's theory (Duato, 1996). Deadlock recovery strategies rely on the fact that deadlock occurrences are rare when the router provides high levels of adaptivity (Anjan and Pinkston, 1997). Instead of using many resources (i.e. virtual channels) to avoid deadlock, we will need lesser resources per router to forward deadlocked messages. However, deadlock detection is not a trivial problem and poor performance may result from either low detection rates or false deadlocks. Misrouting can also be used to avoid deadlock as well as to increase fault tolerance and circumvent congestion areas. However, any non-minimal routing introduces a livelock risk, as a message that is misrouted may be prevented from reaching its destination in a bounded time.

Finally, most router designs avoid starvation by providing a fair arbitration scheme that guarantees a bounded waiting time for any packet requesting an output channel. Note that starvation is the worst-case scenario of network unfairness in which a particular computation node remains unable to access the network resources for an unbounded time limit. Lesser cases of network unfairness will allow different network nodes to inject packets at different rates, resulting in some of the nodes experiencing saturation while other nodes are still able to inject at their full rates. Dally and Towles (2004) stated that network unfairness is caused by unfair arbitration. They cite the chaos router as an example: its arbiter gives priority to packets queued at the router over incoming packets. When routing arbitration is fair, it is expected that the whole network will also be fair in terms of throughput. Thus, the only metric used for measuring network throughput is the number of packets (of flits) delivered per unit of time.

Recent work on limited injection mechanisms has highlighted the presence of network unfairness for non-uniform loads (Izu, Miguel-Alonso and Gregorio, 2005). Such loads caused a non-uniform use of network resources beyond saturation. This paper extends that evaluation by showing network unfairness is present in most router proposals at loads beyond saturation, regardless of their flow control, routing strategy or arbitration policy. Furthermore, we will see that some networks exhibit throughput unfairness even under uniform loads. Such findings question the validity of reported network performance under heavy loads. Furthermore, this work restates the need for better synthetic loads that reflect the behaviour of parallel applications at saturation (Chien and Konstantinidou, 1994).

2 Interconnection Network design

In this section we will briefly describe the main router proposals for k-ary n-cube networks (Duato, Yalamanchili and Ni, 1997). As we aim to prove that network unfairness occurs in most routers, we need to consider a representative set of k-ary n-cube routers, including both oblivious and adaptive routers. In most cases, deadlock management has a significant impact on router architecture and channel utilization. Thus, we will cover the full range of deadlock avoidance methods.

2.1 Static routers

Dimensional order routing (DOR) in a k-ary n-cube network forwards packets in dimensional order: the path from a source node A with coordinates (a_1, .., a_n) to a destination node B = (b_1, .., b_n) will travel in the first dimension to node (b_1,a_2, ..a_n), then on the second dimension to read the node with coordinates (b_1,b_2 …a_n) and so for until reaching the destination node. This strategy is also called oblivious or static routing.

A torus can be seen as a collection of uni-dimensional rings (each row or column in a 2D torus). As nodes travel dimensions in a fix order, it is not possible to form deadlock cycles over multiple rings. However, it is possible to reach a deadlock configuration inside one of these rings.

The channel dependency graph (Dally and Seitz 1987) of any unidirectional ring has a cycle as shown in figure 1.(a). This represents a network in which each node in one ring (for example, row 2 in the +X direction) has full input and output buffers and none of those messages have reached their destination yet. As the next node's buffer is also full, no message will be forwarded; all of them will continue to wait for the next input buffer to become available.

2.1.1 DOR-2vc router

The DOR-2vc router is an oblivious router that divides each physical channel into two virtual channels to avoid deadlock, as illustrated in figure 1(b). Messages that cross the wrap-around link must change from using virtual channel 0 to using virtual channel 1. This eliminates the cyclic dependencies in each unidirectional ring as shown in the channel dependency graph of figure 1(c).

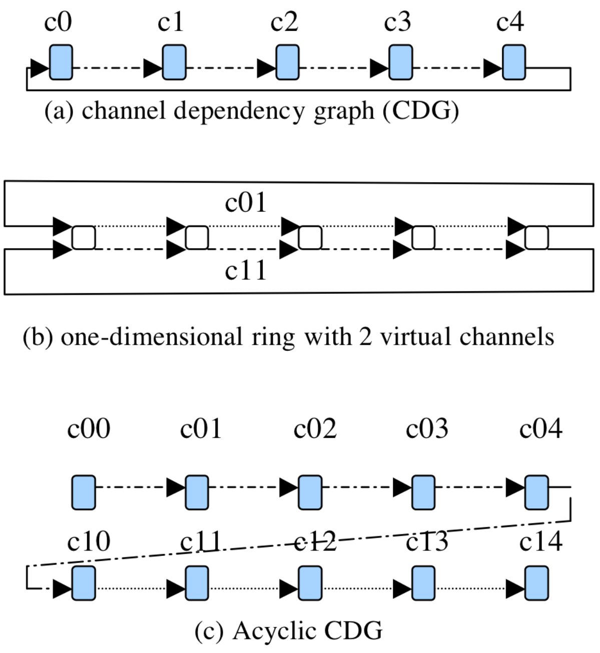

c0
c1
c2
c3
c4
(a) channel dependency graph (CDG)
c01
c11
(b) one-dimensional ring with 2 virtual channels
c00
c01
c02
c03
c04
c10
c11
c12
c13
c14
(c) Acyclic CDG

Figure 1. Breaking the channel dependency cycle using virtual channels

This method was initially proposed for wormhole torus networks (Dally and Seitz 1987) and it has been used extensively as it was the only proven method to avoid deadlock in wormhole networks. Note that this deadlock avoidance method is also applicable to virtual cut-through networks.

2.1.1 DOR-Bubble router

Bubble flow control (BFC) is an extension of VCT flow control that prevents the network interface from filling up its router buffer capacity (Carrion et al, 1997). If the ring is not full, deadlock cannot occur. Figure 2 illustrates the use of bubble flow control in a unidirectional ring.

Packets (shaded queue units) are allowed to move (shaded arrows) from one queue to another inside the ring as per virtual cut-through switching. However, packet injection is only allowed at a given router if there are at least two empty packet buffers in the local queue on top of the VCT restriction. By doing so, we guarantee that, even when multiple nodes inject simultaneously, there will always be at least an empty packet buffer in the ring. That free buffer acts as a bubble, allowing at least one packet to progress. Both packet injection and packet turning from an X ring to a Y ring are subject to BFC. Packets inside a ring move as per VCT.

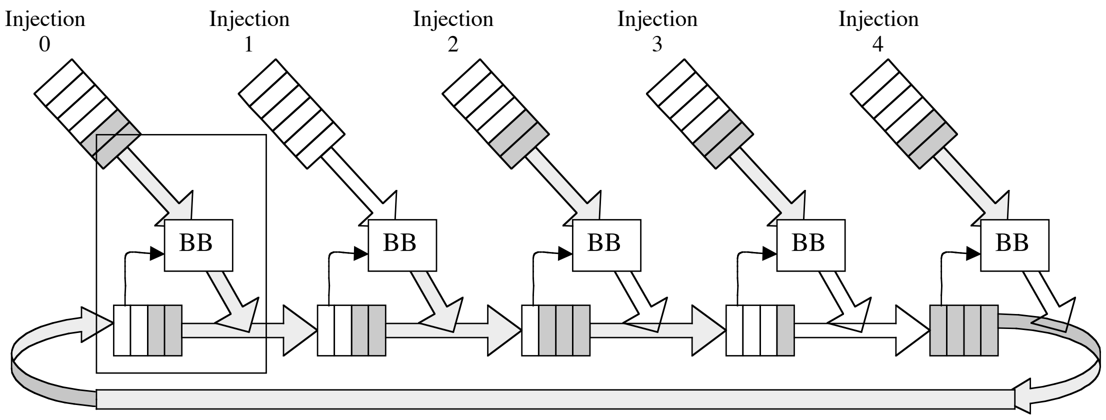


Figure 2.- Deadlock avoidance in a ring using Bubble Flow control.

2.2 Minimal Adaptive routers

Fully adaptive routers allow packets to select any minimal path between source and destination, based on the network status. The packet travels along a default path (probably DOR) but if its next output channel is busy it will change dimension of travel. A packet will block when it cannot progress in *any* dimension. Thus, latency is reduced at medium loads.

As packets may turn in any direction, fully adaptive routing increases the risk of deadlock. Fortunately, Duato's theory (1996) provides a framework on how to built deadlock free fully adaptive networks: it is possible to combine a fully adaptive virtual network with a deadlock-free virtual network so that the latter provides escape paths for any potentially deadlocked packet in the fully adaptive sub-network.

2.2.1 Duato4vc

This router is built by adding two more fully adaptive virtual channels to the DOR-2vc router. Note we can add any number of fully adaptive channels. Adding more virtual lanes may reduce even further head-of-line blocking but at the cost of higher arbitration and crossbar complexity.

This network is similar to that use in some commercial systems such as the Alpha 21364 (Mukherjee et al, 1997) or the Cray T3E (Scott and Thorson, 1996).

2.2.2 Adap-Bubble

This router is built by adding two more fully adaptive virtual channels to the DOR-bubble router. Thus, it uses 3 virtual channels per physical link. There is no restriction to inject packets in the adaptive sub-network and packet can move form escape to adaptive channels as required. Changing from an adaptive channel to an escape one must meet the bubble flow control conditions; for a full description see (Puente et al, 2001). This router design has been implemented in the torus network of the BlueGene/L supercomputer (Blumrich et al, 2003)

2.2.3 Disha4vc

This fully adaptive wormhole router implements a deadlock recovery strategy that forwards a deadlocked packet using a dedicated central buffer per router as described in (Anjan and Pinkston 1997).

Disha is reminiscent of Duato's approach used in the other router as it has two virtual networks -- one susceptible to deadlocks (possibly adaptive) and the other that provides escape routes. However, there are significant differences. Escape paths in Duato's scheme use two virtual channels as per DOR-2vc. However, the escape channel in Disha is a single Deadlock Buffer central to the router. This buffer is shared between neighbouring nodes and, unlike edge buffers, is not dedicated to any path. A packet is assumed to be deadlocked after its blocking time at the node reaches a threshold. The selection of a proper time-out interval is important to obtaining optimum performance. Deadlock feeds upon itself in that if cycles are not broken quickly, more and more

2.3 Non-Minimal adaptive routers

A non-minimal adaptive router allows packet to select output channels that will taken further away from their destination. This allows packets to circumvent faults and/or avoid minimal path congested areas. On the other hand, each packet uses more network resources so that network throughput may be reduced.

2.3.1 The Chaos router

The chaos router is a fully adaptive virtual cut-through router that instead of using virtual channels to avoid deadlock, it relies on misrouting of the blocked packets.

In the absence of congestion packet follow non-minimal paths. The router has one buffer per input channel, and a central queue to store blocked packets, which have not being able to cut-through while the rest of the packet was transmitted. Once the central queue is full, and a packet is blocking the input channel, misrouting is triggered at one of the queued packets is forced to use the available output channel. The packet at the input can then be queued so that the input channel is ready to accept another packet. Thus, a packet whose destination is in a congested area may be forced to take non-minimal paths once or more times. The chaos router solves livelock by randomising the selection of the packet to be misrouted, so it is most unlikely for a given packet to be repeatedly misrouted. For more details please refer to (Bolding, Fulham and Snyder, 1997).

3 Evaluation methodology

Evaluations of architectural proposals need to be carried out during all design stages. Simple functional simulators help us assessing routing algorithms, deadlock avoidance mechanisms, fault-tolerance, etc during the early stages. These simulators do not incorporate all the details required in an actual, hardware-implemented system; however, the most relevant aspects of the design are there, allowing us to check the viability of a proposal—or its drawbacks. In subsequent stages, more detailed simulators or even hardware prototypes can be used to refine the design.

Network performance is reported using two figures: latency (time from packet generation until its delivery) and throughput, which is measured as the number of packets delivered in a given time interval divided by the interval length and the network size. In other words, this is the average load accepted by the network (i.e., the network throughput), which is expected to be even amongst the network nodes. Chien advocated in (Chien and Konstantinidou, 1994) that other throughput measures are needed to reflect relevant throughput characteristic such as fairness and guarantee of throughput. However, most network studies kept on reporting only peak throughput at saturation.

In order to evaluate network fairness we have modified the three simulators described below, so they now measure the number of packets injected per processing node. Using this injection matrix we can calculate not only the average network throughput but also the minimum and maximum throughput per node as suggested in (Dally and Towles, 2004).

3.1 Network simulators

In most comparative studies, the same network simulator is used to compare two or more design alternatives. As the goal of this paper is not to compare routers but to estimate the network unfairness or each router proposal, we have use a range of functional simulators whose source has been made available by their authors, and which are representative of the state of the art in interconnection network simulation. In particular, the chaos router is evaluated using the chaos simulator, Duato4vc and Disha4vc are evaluated using flexsim 1.2 (2005) and the static and adaptive bubble routers are simulated with FSIN (2005). This reflect the way network throughput was measured in the state of the art literature, and provides a wider choice of router implementation details, as per router proposal, instead of the uniformity provided by a single simulator.

All simulators emulate the pipeline stages of a network router each cycle: reception of phits, header decoding and generation of channel request, arbitration and crossbar transmission, virtual channel arbitration and phit transmission. Each simulator has its own set of parameters that allows us to compare different design alternatives, such as input buffer size, the number of virtual channels, the routing policy etc. For example, *flexsim* emulates a range of wormhole routers including dor-2vc, disha and duato. FSIN emulates VCT routers and chaos emulates oblivious wormhole, oblivious cut-through and chaos. In most cases we have use the default parameter values assumed by the simulator. All of them allow us to choose the network size, which we set to a 16x16 torus network (with full-duplex links). We set the packet length to be 16 flits (or phits if it is a VCT router[1]) and in most cases the input buffer capacity is for two packets (except chaos which has single-packet buffers).

For practical reasons, most performance studies of interconnection networks are carried out using synthetic traffic. Each processing node is modelled as an independent traffic source, which generates packet following a Bernoulli (or sometimes Poisson) distribution with a parameter that depends on the applied load. All simulators provide standard destination functions such as random, a range of permutations (transpose, bit reversal, perfect shuffle etc) and random with hot-spots.

4 Network Evaluation

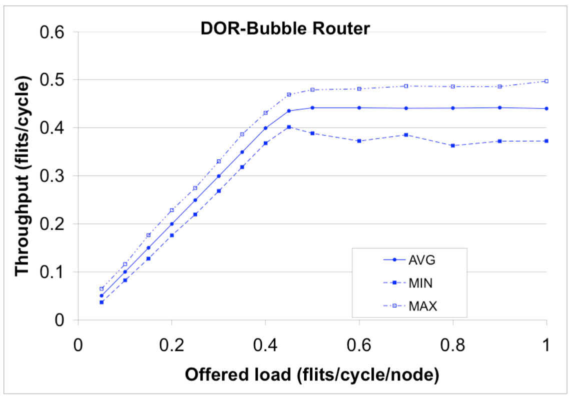
This section presents throughput results under both uniform and non-uniform loads for each of the routers under consideration. This results complement the average throughput values reported in the literature.

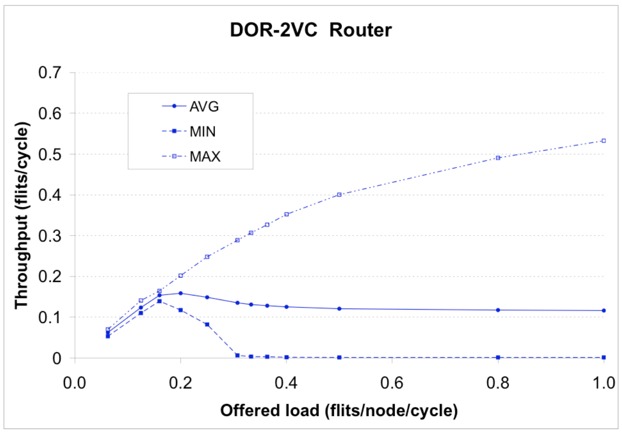
4.1 Random Load

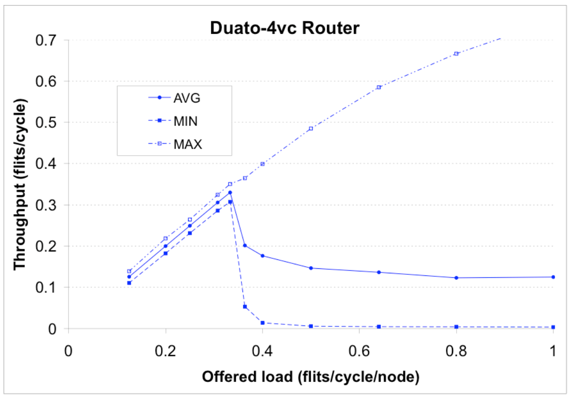
Figures 3 and 4 show the range on node throughput values versus load for a range of router designs. Under random traffic most networks exhibit minor variations between the minimum and maximum throughput experience by any given node. These variations are probably due to the minor traffic fluctuation which impact on their injection rates. Note that random traffic makes an even use of the physical channels. In most routers, channel utilization is balanced amongst the VCs, as packets are free to move from one virtual channel or lane to another.

Both DOR-Bubble and Chaos give preference to transit packets over new injections, so that they cause starvation. However, under random traffic they seem to be reasonably fair at very high loads. On the contrary, both DOR-2vc and Duato4vc exhibit significant network unfairness for load beyond saturation. The main difference for this type or routers is their unbalanced use of the oblivious virtual channels as reported in (Bolding, 1992).

[1] Flit stands for FLow control unIT, which in VCT is a packet; phit is the physical unit sent in one cycle.







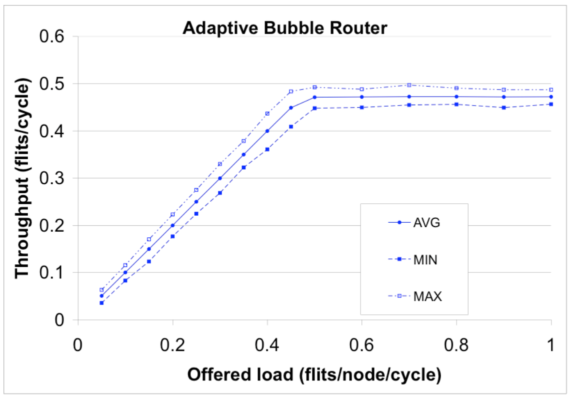
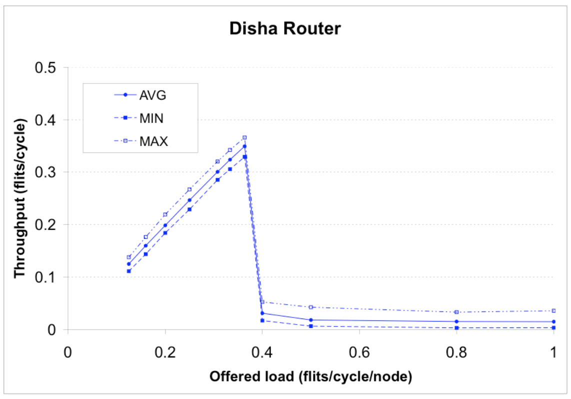


Figure 3. Minimum, maximum and average node throughput versus offered load for a range of deadlock-free minimal 16x16 torus networks under random traffic.



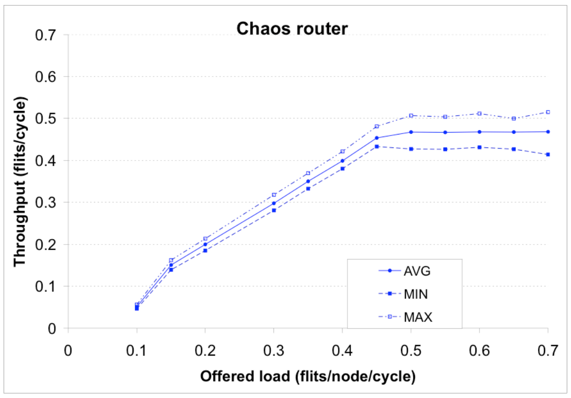


Figure 4. Minimum, maximum and average node throughput versus offered load for a Chaos (top) and a Disha (bottom) 16x16 torus network under random traffic.

As flexsim 1.2 does not provides virtual channel utilization maps we did run the equivalent DOR-2vc under FSIN, selecting an optimised version on Dally's deadlock-free routing function so that packets not crossing the wrap-around link are injected in any of the virtual channels. Figure 5 shows the channel utilization map for the channel +X for both Dor-Bubble and DOR-2vc under 10% of the maximum network load.

We could see that the Bubble router exhibits a balanced used of the network channels while the Dor-2vc router exhibits, in spite of the optimisation, a quite unbalanced use of its two virtual channels. This is because by selecting a fixed link as the wrap-around that limits virtual channel utilization, the network symmetry is broken: nodes close to the link are forced to use VC0.

Note that a high loads many packets will blocked and resort to use the escape sub-network, including new packet at the injection ports. In other words, a packet at injection will encounter different channel utilization (for the VC selected by the routing function) depending of its network location. The Adaptive Bubble router is slightly fairer that its oblivious counterpart, as the additional virtual channels do not disadvantage new packets. Disha is quite fair as well but suffers significant degradation at loads beyond saturation.

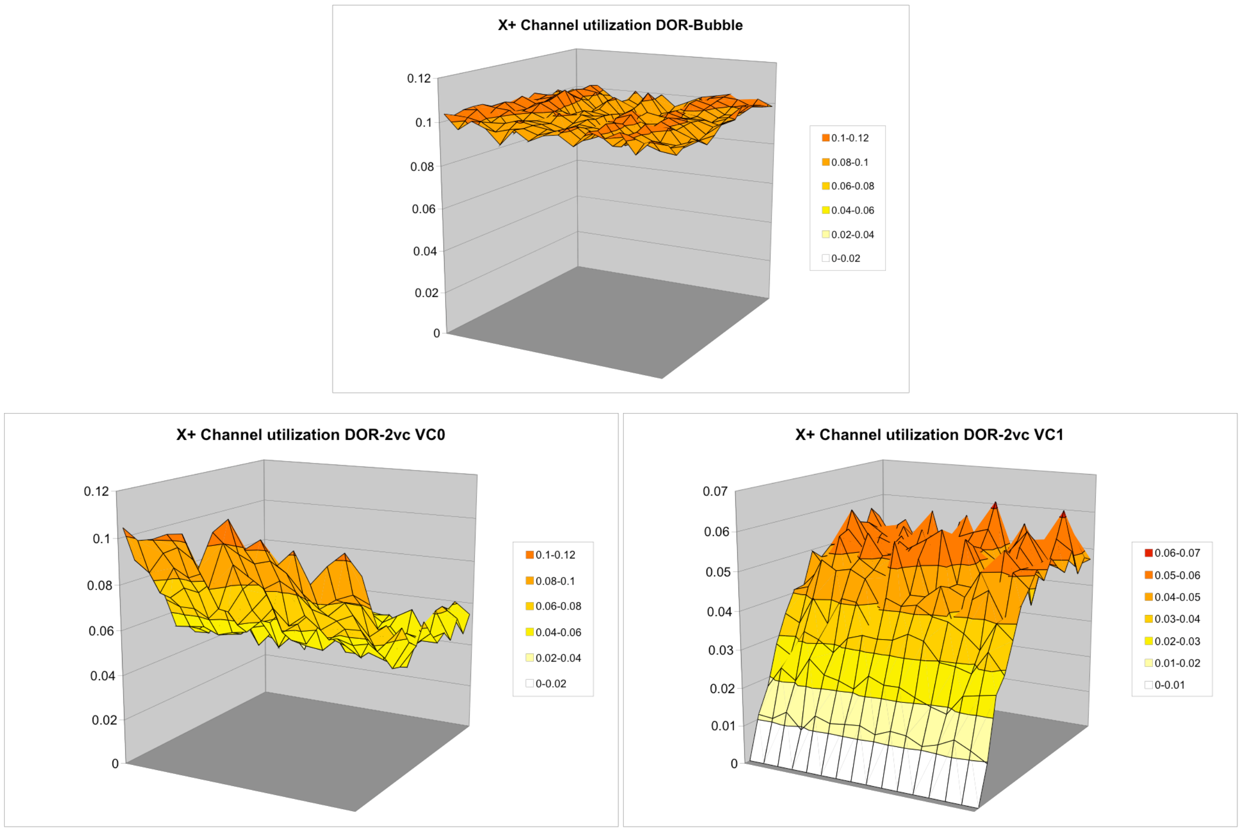


Figure 5. Channel utilization for the two oblivious routers under 10% random offered load.

4.2 Non-Uniform Loads

All permutation patterns made a very unbalanced usage of network resources under DOR routing. Adaptive routing addresses this issue by exploiting multiple paths but most patterns still exhibit uneven channel utilization. For example, the transpose pattern builds congestion amongst the network diagonals and the bit-reversal permutation does put pressure on the network bisection. In this section we will limit the discussion to the transpose permutation, but throughput unfairness is significant for any non-uniform traffic pattern.

Router	Minimum	Average	Maximum
DOR-2vc	0.0014	0.14218	0.59792
DOR-Bubble-	0.0001	0.1334	0.4630
Duato4vc	0.0017	0.2557	0.7065
Adap-Bubble	0.0988	0.2339	0.6508
Chaos	0.0008	0.2067	0.4796
Disha4vc	0.0019	0.2752	0.6632

Table 1. Minimum, maximum and average node throughput (flits/cycle) for a 16x16 tours network with transpose permutation pattern at 0.8 flits/cycle/node offered load.

4.1.1 Transpose

Table 1 summarizes the average, minimum and maximum node throughput in flits/cycle. We can see all networks exhibit high levels of throughput unfairness. To explain these values we need to look at the channel utilization for each router. We will start first showing the values for the chaos router, which is simpler as it does not have virtual channels.

Figure 6 shows the distribution of X+ channel utilization in the Chaos router both below and above saturation. At loads below saturation, the injection matrix is flat as all nodes are able to inject their packets. Note that the nodes in the diagonal don't send messages through the network as the destinations are their own nodes, hence their value is zero. Any other node is injecting approximately 800 packets. This matrix is similar regardless of the router design chosen, provided the load is below its saturation point.

For loads above saturation we observe there is a noticeable change in the channel utilization, which is due to the use of misrouting. At high loads, the routers around the congested diagonal will fill their central queues and force many packets to misroute. This causes the increase in channel utilization close to the diagonal.

The injection matrix shows a direct co-relation between high utilization channels and low number of injected packets. In other words, a peak in channel utilization results in a valley in the injection matrix.

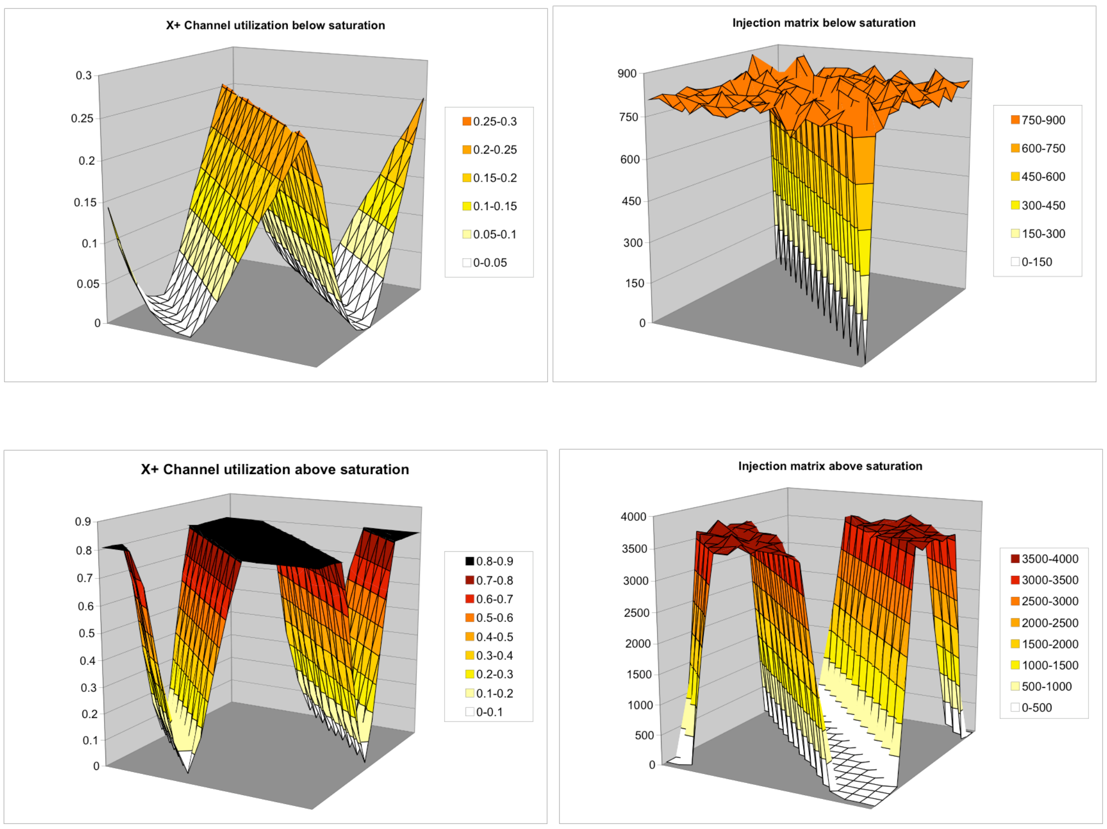


Figure 6. Channel utilization (left) and distribution of the injection rates for a 16x16 chaos network under transpose traffic pattern for a 0.8 flits/cycle/node offered load.

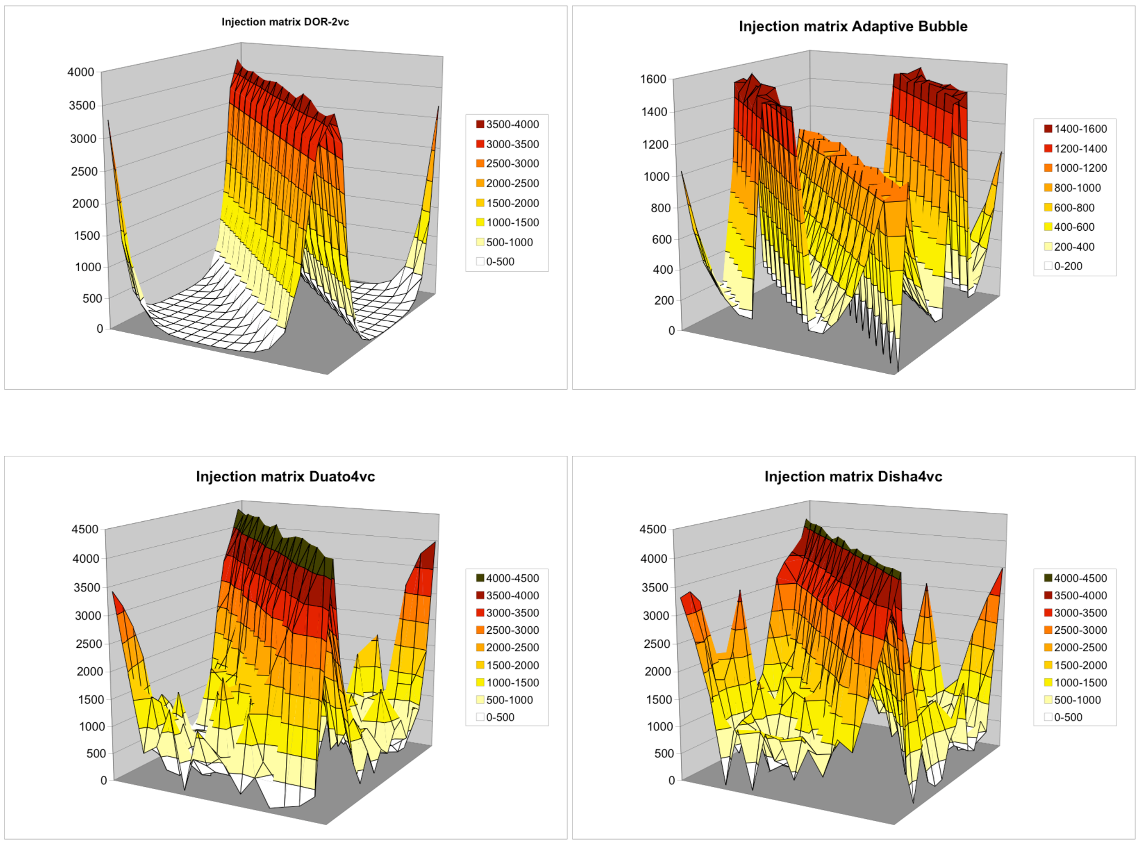


Figure 7. Injection matrix for a 16x16 network under transpose traffic (0.8 flits/cycle/node offered load) with a range of router alternatives.

Similarly, a valley in the channel distribution graph turns into a peak in the injection matrix. Note that some nodes inject well above the theoretical limit of 0.5 flits/cycle as they are in low utilization areas, so they are able to inject at their wish. This additional traffic increases congestion in the diagonal and further reduces the injection rates of the nodes close to the diagonal.

This relation occurs in all routers, although the injection matrix varies depending on the ability of the router to distribute the packets amongst its set of virtual channels. Figure 7 shows the injection matrix distribution at 80% load for other network alternatives. All of them exhibit large levels of unfairness, as described in Table 1, Note that all the wormhole routers have a similar unfairness pattern, which is related by the default paths and arbitration used in the flexsim simulator. Is out of the scope of this paper to analyse the differences amongst them.

We could see that the router architecture has a negligible impact on network fairness. In fact, the chaos router proves to be one of the more fair alternatives for the transpose traffic pattern. Besides, design choices such as increasing the buffer size or the number of virtual channels do increase peak throughput but do not alter the channel utilization patterns.

All the above results link network fairness to the balanced usage of the network resources and not to the arbitration policy as suggested in (Dally and Towles, 2004). Similar uneven figures are obtained for other permutation patterns such as bit-reversal and perfect shuffle.

5 Summary

This paper has shown that network throughput seen by each computing node at saturated loads varies with node location. This indicates that reported values of *average* network performance at heavy loads, beyond its saturation point, are to not be relied upon.

All network designs exhibit significant network unfairness under non-uniform loads. Furthermore, duato4vc, a popular router design implemented in many commercial systems, exhibits unfairness under uniform random traffic. We can conclude that fair arbitration is not sufficient to guarantee network fairness; in that case, the node injection rate depends on the level of activity of its router. Thus, network fairness relies on the ability of the network to balance channel utilization amongst its routers.

Network unfairness is not desirable in terms of application performance. A tightly coupled application in which there is a high level of data exchange amongst the nodes will keep them working at the same pace. A loosely coupled application may allow some nodes to race ahead of the pack, only to wait later at some synchronization barrier, reducing the overlap between computation and communication. Further work is needed to explore if network fairness will bring significant gains to application performance.

6 Acknowledgements

I would like to thank my colleagues Jose and J.A. whose collaborative work lead us to consider fairness as an issue for the adaptive bubble router.

Thanks also to the three network research groups for making their simulation code available to others, so that I could save many hours on writing code to emulate their proposals.

7 References

K. V. Anjan, Timothy Mark Pinkston: An Efficient, Fully Adaptive Deadlock Recovery Scheme: DISHA. Proceedings of the 22nd Annual International Symposium on Computer Architecture, ISCA '95 pp. 201-210,

Anjan K.V. and T.M. Pinkston (1997). *An Efficient, Fully Adaptive Deadlock Recovery Scheme: Disha.* Proceedings of the 22nd Annual International Symposium on Computer Architecture, pages 201--210,

N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network. IEEE Micro, 15(1):29-36, February 1995

Kevin Bolding (1992). *Non-Uniformities Introduced by Virtual Channel Deadlock Prevention*, University of Washington, Technical Report UW-CSE-92-07-07.

Bolding, M. L. Fulgham, L. Snyder*, The Case for Chaotic Adaptive Routing*. IEEE Trans. Computers 46(12): 1281-1291 (1997).

M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burrow, T. Takken, P. Vranas. *Design and Analysis of the BlueGene/L Torus Interconnection Network* IBM Research Report RC23025 (W0312-022) December 3, 2003.

C. Carrión, R. Beivide, J.A. Gregorio and F. Vallejo (1997), *A Flow Control Mechanism to Prevent Message Deadlock in k-ary n-cube Networks*, Proceedings of the Fourth International Conference on High Performance Computing (HiPC'97). Bangalore, India.

A. A. Chien and M. Konstantinidou (1994) "Workload and performance metrics for evaluating parallel interconnects," IEEE Computer Architecture Technical Committee Newsletter, Summer-Fall 1994 pp. 23 - 27,

Chaos simulator (1996) from the Chaos Project group http://wotug.ukc.ac.uk/parallel/simulation/communications/chaos/

W.J. Dally, B. Towles (2004). Principles and Practices of Interconnection Networks. Morgan-Kaufmann, 2004.

W.J. Dally, C.L. Seitz (1987): The Torus Routing Chip Distributed Computing 1:187-196.

J. Duato (1996). "A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks". IEEE Trans. on Parallel and Distributed Systems, **7**: 841-854.

J. Duato, S. Yalamanchili and L. Ni (1997) Interconnection Networks: an engineering Approach, IEEE Computer Society Press.

FSIN (2005) functional simulator for interconnection networks from the Parallel Technology Group at http://www.sc.ehu.es/acwmialj/ptech/index.html

C. Izu, J. Miguel-Alonso and J.A. Gregorio (2005), Evaluation of Interconnection Network Performance Under Heavy Non-uniform Loads, ICA3PP 2005, LNCS 3719, pp. 396 –405, 2005.

Konstantinidou S. and Snyder L.(1990) The chaos router: A practical application of randomization in network routing. 2nd Ann. Symp. on Parallel Algorithms and Architectures SPAA'90 pp. 21-30.

S. Mukherjee, P. Bannon, S. Lang, A. Spink and David Webb (1997), "The Alpha 21364 Network Architecture", IEEE Micro **21**:26-35.

F. Petrini, W. chun Feng, A. Hoisie, S. Coll, and E. Frachtenberg (2002) The quadrics network: High-performance clustering technology. IEEE Micro, **22**(1):46—57.

Pinkston T. M. and Warnakulasuriya S (1997) On Deadlock in Interconnection Networks. Proceedings 24th Int. Symposium Computer Architecture, ISCA'97, Denver.

V. Puente, C. Izu, J.A. Gregorio, R. Beivide, and F. Vallejo (2001), The Adaptive Bubble router, Journal on Parallel and Distributed Computing, 61(9) 1180-1208.

FlexSim1.2, from the SMART group at the U. of Southern California., accessed 22 August 2005, http://ceng.usc.edu/smart/FlexSim/flexsim.html

S. L. Scott and G. Thorson (1996), The Cray T3E networks: adaptive routing in a high performance 3D torus, Proc. of Hot Interconnects IV.

A JMX Toolkit for Merging Network Management Systems

Feng Lu **Kris Bubendorfer**

School of Mathematical and Computing Sciences
Victoria University of Wellington,
P. O. Box 600 Wellington, New Zealand,
Email: `Feng.Lu@mcs.vuw.ac.nz, kris@mcs.vuw.ac.nz`

Abstract

The ever increasing size of networks has resulted in a corresponding escalation of administration costs and lengthy deployment cycles. Clearly, more scalable and flexible network management systems are required to replace existing centralised services. The work described in this paper forms part of a new network management system that fuses dynamic extensibility, Java Management Extension (JMX), and mobile agents. The primary focus is on integration with the many widely deployed legacy SNMP-based network management systems. One of the primary contributions is the design of a generic SNMP adaptor to enable JMX compliant agents to be accessed by SNMP-based management applications. A set of SNMP APIs have been developed to support the development of the SNMP adaptor. A number of other tools have been developed to support the SNMP adaptor, these include: a Management Information Base (MIB) compiler that automatically generates MBeans representing a given SNMP MIB; and a SNMP proxy service to allow non-SNMP management applications to access the SNMP agent using a variety of protocols.

Keywords: JMX, Network Management, SNMP

1 Introduction

Traditional network management (NM) approaches, such as SNMP (Simple Network Management Protocol) and CMIP (Common Management Information Protocol), are based on a static centralised management platform: a centralised manager acting as client controls the entire network through agents which reside in each network node acting as server. Agents are responsible for monitoring and controlling managed objects in the network. The manager has the responsibility of collecting data from agents, interpreting that data and directing the agents (Yemini et al. 1991). However, with the rapid growth of networks, such approaches are no longer suitable as the increasing complexity of these systems results in high administration costs and long deployment cycles. The need for more scalable and flexible network management approaches is leading to greater decentralisation (Simoes 1999).

Dynamic extensibility is one of the solutions that has been used to support the distributed network management model. The extensible agent can add or delete managed objects and management services at run time upon requests by other management entities. Recently, Sun Microsystems introduced a set of standards to equip Java with an extensible agent model for distributed management, known as Java Management Extension (JMX).

JMX aims to achieve the goal of scalable, distributed network management (JMX1.2 2002). Its support for mobile code enables the transfer of lightweight applications to management agents at runtime, delegates the management tasks from a centralised manager to management agents distributed around the network, and place the management tasks closer to the management data. This reduces network traffic and increases scalability (Lange et al. 1999). Also, the JMX component based architecture allows each JMX resource or service to be plugged into or removed from the management agent dynamically, depending on the runtime network requirements. This means that a JMX based implementation can scale from small handhold devices to large telecommunications switches.

However, legacy management systems are still widely deployed. Network managers have to rely on the legacy management protocol to access the network resources in the heterogeneous network environment. Therfore, a JMX based solution needs to coexist with and integrate with traditional network management systems, like SNMP, instead of replacing them. It is attractive and cost-efficient to develop and deploy a distributed management system using JMX that can cooperate with deployed legacy system. This interoperability can be achieved by equipping the JMX-based solution with SNMP capability. Without it, the JMX-based solution will not become a general solution for distributed network management.

The research efforts presented in this paper focus on the the integration of JMX with traditional SNMP-based network management systems. One of the primary contributions is the design of a generic SNMP adaptor to enable JMX compliant agents to be accessed by SNMP-based management applications. A set of SNMP APIs have been developed to support the development of the SNMP adaptor. A number of other tools have been developed to support the SNMP adaptor, these include: a Management Information Base (MIB) compiler that automatically generates MBeans representing a given SNMP MIB; and a SNMP proxy service to allow non-SNMP management applications to access the SNMP agent using a variety of protocols.

2 Background

The majority of deployed network management systems utilise a centralised approach, where the management application periodically accesses the data collected by a set of software modules on network devices. The software modules on network devices are mainly concerned with information gathering and simple calculation, while the management application handles decision making and higher level functions. The centralised approach is driven by two assumptions (Goldszmidt 1993):

- Network devices lack resources to execute complex computational tasks.
- Management data and functions are relatively simple.

The Simple Network Management Protocol is the dominant protocol in existing managed systems. The protocol is designed to be an easily implemented, basic network management tool. The SNMP set of standards defines an information management model along with a protocol for the exchange of the information between a managed device with an SNMP agent and an SNMP manager. International Standard Organisation (ISO10165-1 1993) presents another general management information model of OSI systems management information. The ISO model has a similar approach to the SNMP management model, but differs in the way it operates.

The rapid expansion of networks has resulted in real network management problems that can't be adequately addressed (Meyer et al. 1995). Also, the computational capability of network devices has increased. The increase in the capability of managed devices has made it possible to distribute complex computations and significant duties to the managed devices (Puliafito et al. 2000). Research on the decentralised approaches to network management began as early as SNMPv1's RMON (Remote Network Monitoring) MIB. The core of RMON are the remote monitors, that take responsibility for the collection and analysis of statistical information on network traffic and device status on sub-networks. The remote monitors report only significant information to the SNMP managers. The enhanced SNMPv2 provides a manager-to-manager (M2M) MIB to support a hierarchical management architecture. Similar to the RMON, the M2M allows a sub-manager to function as a remote monitor for a sub-network. The latest SNMPv3 management framework makes it possible to develop a set of distributed entities, composed of several interacting modules. However, the SNMP management framework does not specifically address distribute network management. Instead, the IETF DIAMAN working group proposes a distributed management architecture based on the SNMP management framework (DISMAN 1996).

On the other hand, Yemini and Goldszmidt (Yemini et al. 1991) proposed the Management by Delegation (MbD) model for distributed network management. The fundamental idea behind this approach is to dynamically distribute management functions amongst management entities. The MbD model is based on the technology "code mobility". It moves the code, describing management functions, closer to the data they process. Moving code is more efficient if the amount of data that needs to be transferred is larger, and reduces the total amount of network management traffic (Schonwalder 1997). The MbD model consist of three parts: a delegation protocol, a delegation language and an agent. The delegation protocol is used to communicate between managers and agents. The delegation protocol enables the manager to transfer the delegation code, to control the behaviour of the delegation code (execute, suspend and stop etc.), and to retrieve the results of the execution. The delegation language is used to write management functions, that can be executed at runtime. Several different languages have been in different research prototypes ranging from high-level interpreted languages to low-level stack-oriented languages (Schonwalder 1997). A MbD agent acts in both an agent role and a manager role. To managers requesting information from the MbD Manager, it is an agent, while to those agents it queries, it is a manager. The MbD agent provides the services to parse and execute received delegation code. It also provides the interfaces that enable the remote manager to control the execution of the delegation and retrieve the results. In addition, the MbD agent can delegate its management functions to other MbD agents.

Dynamic extensibility has been used to support the MbD model for distributed network management. Extensible agents are MbD agent that can dynamically add or delete managed objects upon the requests from other management entities. The early extensible agent model is based on the SNMP framework with a distributed MIB consisting of a static MIB residing in the master agent and several temporary MIB dynamically registered by subagents.

This paper relates our experience in designing and implementing a JMX based network management toolkit.

3 JMX Architecture

The JMX specification provides a framework for a distributed management model based on manageable resources, dynamically extensible agents and distributed management services as shown in Figure 1. The JMX architecture is separated into three layers: the instrumentation level, the agent level and the distributed services level.

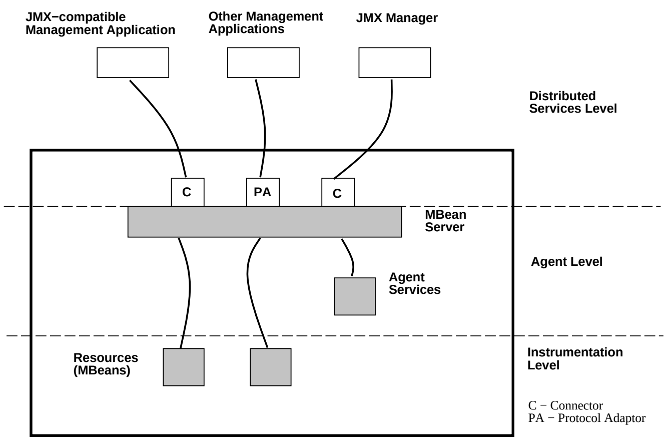


Figure 1: *JMX Architecture*

3.1 Instrumentation Level

The instrumentation level provides a specification for implementing JMX manageable resources. A resource can be an application, a device, or the implementation of a service. A JMX manageable resource must comply with the MBean standard defined in the JMX specification, and may be dynamically added to or removed from the JMX agent. MBeans encapsulate manageable objects as attributes and operations through their public methods, and utilise design patterns to expose them to management applications (JMX1.2 2002).

There are two kind of MBean. Standard MBeans provide a static management interface, which is fixed at compile time and is invoked by reflection. The standard MBeans' interfaces are made up of the methods for reading and writing attributes and for invoking operations. The design pattern followed by a standard MBean is derived from the JavaBeans component model (JavaBeans 1999). In this design pattern, attributes are exposed through the getter and setter methods in the MBeans' interface. Attributes may be read-only, write-only or read-write. The return value or arguments of methods for attributes must conform to the data type of attributes. Operations are exposed by the methods other than getter and setter in the MBeans' interface. They can be defined with any number of arguments with any data types.

```
public interface DynamicMBean {

    public Object getAttribute(String attribute)
      throws AttributeNotFoundException, MBeanException, ReflectionException;

    public AttributeList getAttributes(String[] attributes);

    public MBeanInfo getMBeanInfo();

    public Object invoke(String actionName, Object[] params, String[] signature)
      throws MBeanException, ReflectionException;

    public void setAttribute(Attribute attribute)
      throws AttributeNotFoundException,InvalidAttributeValueException,
             MBeanException, ReflectionException;

    public AttributeList setAttributes(AttributeList attributes);
}
```

Figure 2: *DynamicMBean Interface*

Dynamic MBeans conform to a specific interface that exposes the management interface at runtime. Unlike standard MBeans, dynamic MBeans do not have getter or setter methods for each attribute and operation. Instead, the *DynamicMBean* interface is defined to provide generic method for getting or setting an attribute and for invoking an operation. As shown in Figure 2, the *getMBeanInfo* method defined in the *DynamicMBean* interface returns an object which contains meta information about the MBean's attributes, operations and notifications that may be emitted by the MBean.

Using this meta information, management applications can access the MBean's attributes and invoke the MBean's operations through generic methods defined by the *DynamicMBean* interface. Compared with standard MBeans, dynamic MBeans provide a more flexible way to instrument resources and make it simple to instrument existing JMX incompatible resources (legacy management resources, etc.).

Dynamic MBeans can be further refined into two useful specialisations:

- An open MBean is a dynamic MBean that relies on a small, predefined set of universal Java Types to describe managed objects. It is useful where a management application and agent do not share application-specific data types.

- The model MBean, is a generic configurable management template for managed resources. Model MBeans can be used to instrument almost any resources rapidly.

3.2 Agent Level

The agent level provides a specification for implementing the JMX agents that control the MBean resources and make them available to management applications. A JMX agent consists of a MBean server, a set of agent services, and at least one communication protocol adaptor or connector, see section 3.3. The MBean server acts as a central registry for MBeans managed by the agent. Only registered MBeans may be accessed from outside of the MBean server. The MBean server provides a set of interfaces to manipulate MBeans. All management requests are handled by the MBean server, which dispatches them to the appropriate MBean. Through the MBean server, management applications may: register or deregister MBeans, browse and query MBeans, discover the management interface of MBeans, read or write the values of MBeans' attributes, invoke the operations exposed by MBeans, and register and deregister notification listeners for MBeans.

JMX agent services are also MBeans that provide services for other MBeans or management applications. There are four standard services defined in the JMX specification: Dynamic Loading Service, Monitoring Service, Timer Service and Relation Service. Dynamic Loading Service allows the agent to instantiate MBeans using Java classes and native libraries dynamically downloaded from the network. Monitoring Service notifies its listeners on certain conditions or events. Timer Service sends notifications at predetermined intervals and acts as a scheduler. Relation Service defines associations between MBeans.

3.3 Distributed Services Level

The distributed services level defines management interfaces and components that allow remote management applications to perform operations on agents through different protocol adaptors and connectors. Both protocol adaptors and connectors use the services of the MBean server to apply the management operations they receive to the target MBeans, and to forward notifications, such as an attribute change notification, to management applications. Both protocol adaptors and connectors should preferably be implemented as MBeans. This offers greater flexibility to their operation as they can be activated or deactivated through any of the other available adaptors or connectors.

There are two main differences between protocol adaptors and connectors, though they are similar in terms of functionality:

- Management applications that connect to protocol adaptors access the JMX agent through operations defined by the given protocol, and the operations are then received by protocol adaptors and are mapped to those of the MBean server through protocol adaptors; whereas connectors provide a higher level view for the JMX agent through the local representation of the MBean server. The remote management applications using connectors may access the JMX agent as if it were local.

- Management applications that connect to protocol adaptors are usually tied to a given protocol, whereas management applications which use connectors may use different protocols as long as corresponding connectors are provided.

4 SNMP Adaptor

The SNMP adaptor makes the JMX agent accessible from legacy SNMP managers. The SNMP adaptor emulates the standard SNMP agent, and is configured dynamically to provide mappings between SNMP and MBeans and JMX Notifications via XML mapping files. As shown in Figure 3, the SNMP adaptor consists of a SNMP protocol engine and a MIB registry. The SNMP protocol engine is used to receive and parse SNMP messages to determine the type of request and the Object Identifier (OID) of the MIB object. The engine queries the MIB registry and gets the proxy for the MBean object identified by the OID. The engine then invokes the appropriate access function on the proxy, which will forward the invocation to the appropriate MBean object registered with the MBean server. The notification listener receives the notifications, which the SNMP adaptor is interested in, and forwards them to the SNMP protocol engine. The SNMP protocol engine generates the corresponding SNMP trap message.

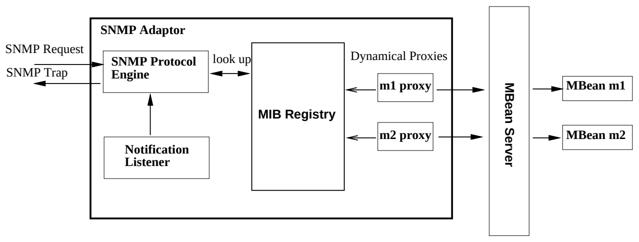


Figure 3: *SNMP Adaptor*

4.1 SNMP Protocol Engine

The SNMP protocol engine is built on top of the JoeSNMP API (OpenNMS 2002) and supports SNMP protocol versions V1, V2c. It consists of several components: transport layer, message dispatcher, message handler and trap generator. These components interact with each other to facilitate communications between the SNMP manager and the SNMP adaptor. Figure 4 describes how the SNMP message is handled by the SNMP engine.

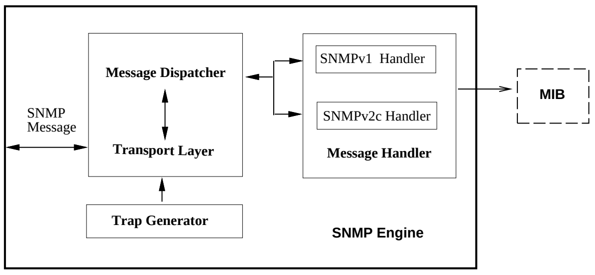


Figure 4: *SNMP Protocol Engine*

1. The SNMP request message is received by the transport layer, and then is forwarded to the message dispatcher. The current transport layer supports UDP.
2. The message dispatcher parses the SNMP message to determine the SNMP version and to extract the Protocol Data Unit (PDU) from the message. Then, it forwards the extracted PDU to the appropriate message handler.
3. There are two message handlers, the SNMPv1 handler and SNMPv2c handlers which are responsible for the corresponding version's SNMP message. The message handler parses the PDU to determine the PDU type and the OIDs of the required MIB objects. The message handler then looks up the MIB Registry to get the required MIB objects.
4. The message handler invokes the appropriate access method on the MIB objects, and then constructs a response message with the new values of the MIB objects.
5. The response message is returned back to the message dispatcher and then is forwarded to the transport layer.
6. The transport layer returns the response message. The transport layer also forwards SNMP trap messages to registered SNMP managers.

4.2 MIB Registry

The MIB registry organises MBean proxies into a SNMP OID tree structure. Figure 5 shows the class hierarchy for the MIB objects in the MIB registry. These objects are organised as several *MIBGroup* objects. A MIBGroup object can not contain other MIBGroup objects. The managed objects in the MIBGroup are represented as *MIBLeafProxy* objects or *MIBTableProxy* objects in terms of the node type.

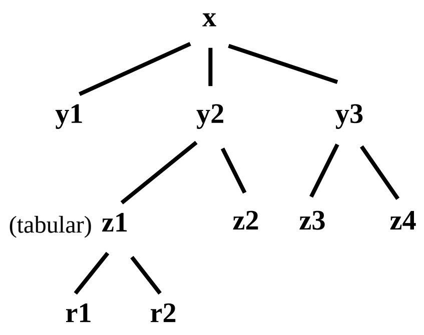


Figure 6: *MIBGroup Example*

For instance in Figure 6, the node `x` has three child nodes: `y1`, `y2` and `y3`. The node `y1` is a leaf node. The node `y2` has two child nodes: `z1` (tabular node) and `z2` (leaf node). The node `y3` also has two child nodes: `z3` (leaf node) and `z4` (leaf node). This OID tree can be represented as follows:

- *MIBGroup* `x` contains *MIBLeafProxy* `y1`
- *MIBGroup* `y2` contains *MIBTableProxy* `z1` and *MIBLeafProxy* `z2`
- *MIBGroup* `y3` contains *MIBLeafProxy* `z3` and *MIBLeafProxy* `z4`

Referring back to Figure 5, the *MIBEntry* abstract class describes the basic structure for a managed object. It contains the attribute *oid* which is used to identify the managed object. It also defines three abstract methods *getRequest*, *getNextRequest* and *setRequest* to handle three primitive SNMP actions: *GET*, *GETNEXT* and *SET*. Both the *MIBLeaf* class and the *MIBTable* class are sub class of the *MIBEntry* abstract class.

A *MIBLeaf* class represents a scalar type managed object, but it also can represent a columnar object of a SNMP table. A columnar object defines the behaviour of managed object instances in a particular column of a SNMP table (Agent++ 2000). The *MIBLeaf* object contains an attribute *value* which represents the instance of the managed object. The *MIBLeafProxy* class extends the *MIBleaf* class with two additional attributes: *mbeanName* and *attribute*. The *mbeanName* represents the object name of the target MBean object, and the *attribute* represents one of the attributes of this MBean object. With these two attributes, the *MIBLeafProxy* object acts as a proxy for the MBean object, and maps the SNMP actions to the appropriate methods on the JMX agent. For instance, when the methods *getValue* or *setValue* is invoked, the *MIBLeafProxy* object invoke the method

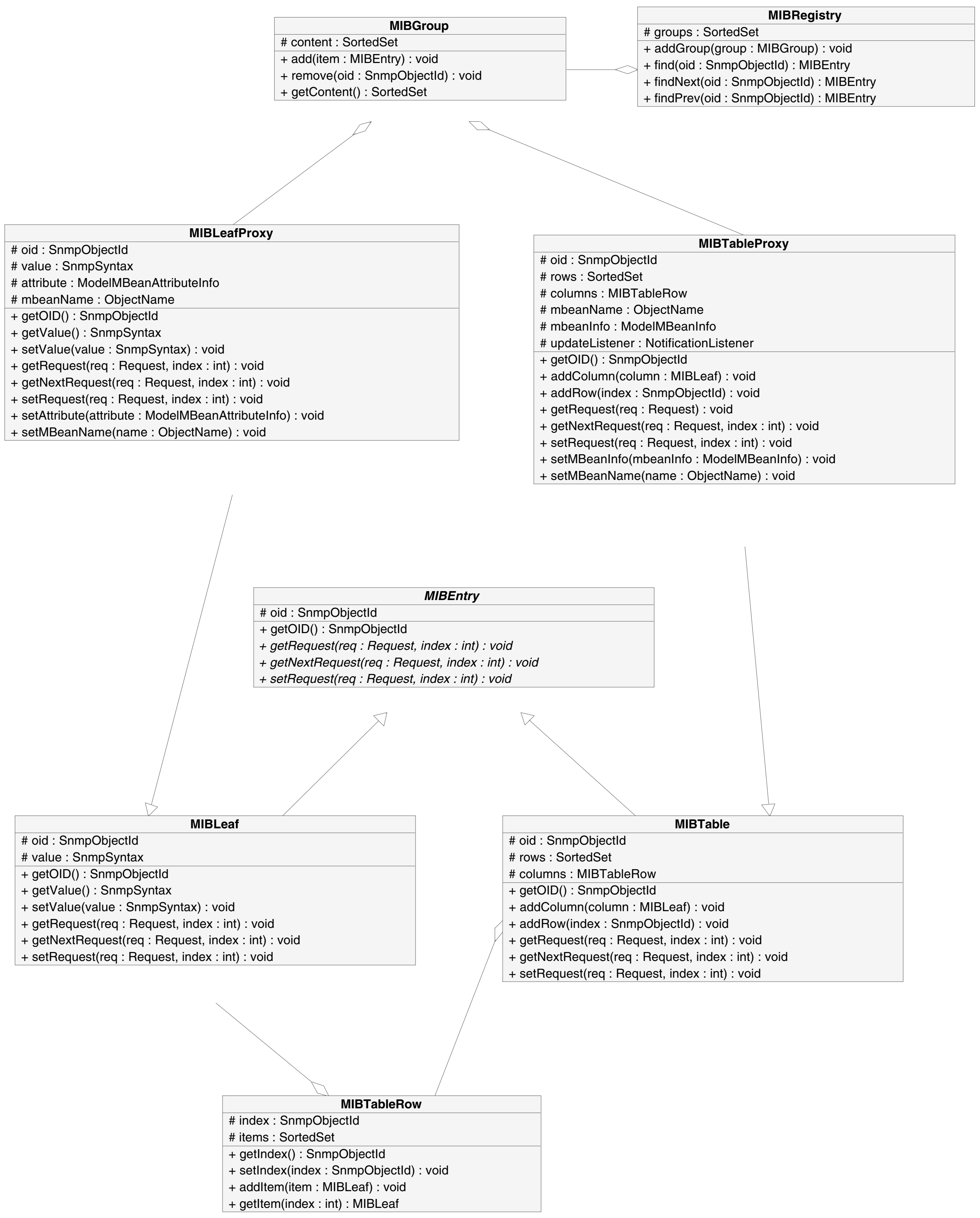

MIBGroup
# content : SortedSet
+ add(item : MIBEntry) : void
+ remove(oid : SnmpObjectId) : void
+ getContent() : SortedSet
MIBRegistry
# groups : SortedSet
+ addGroup(group : MIBGroup) : void
+ find(oid : SnmpObjectId) : MIBEntry
+ findNext(oid : SnmpObjectId) : MIBEntry
+ findPrev(oid : SnmpObjectId) : MIBEntry
MIBLeafProxy
# oid : SnmpObjectId
# value : SnmpSyntax
# attribute : ModelMBeanAttributeInfo
# mbeanName : ObjectName
+ getOID() : SnmpObjectId
+ getValue() : SnmpSyntax
+ setValue(value : SnmpSyntax) : void
+ getRequest(req : Request, index : int) : void
+ getNextRequest(req : Request, index : int) : void
+ setRequest(req : Request, index : int) : void
+ setAttribute(attribute : ModelMBeanAttributeInfo) : void
+ setMBeanName(name : ObjectName) : void
MIBTableProxy
# oid : SnmpObjectId
# rows : SortedSet
# columns : MIBTableRow
# mbeanName : ObjectName
# mbeanInfo : ModelMBeanInfo
# updateListener : NotificationListener
+ getOID() : SnmpObjectId
+ addColumn(column : MIBLeaf) : void
+ addRow(index : SnmpObjectId) : void
+ getRequest(req : Request) : void
+ getNextRequest(req : Request, index : int) : void
+ setRequest(req : Request, index : int) : void
+ setMBeanInfo(mbeanInfo : ModelMBeanInfo) : void
+ setMBeanName(name : ObjectName) : void
MIBEntry
# oid : SnmpObjectId
+ getOID() : SnmpObjectId
+ getRequest(req : Request, index : int) : void
+ getNextRequest(req : Request, index : int) : void
+ setRequest(req : Request, index : int) : void
MIBLeaf
# oid : SnmpObjectId
# value : SnmpSyntax
+ getOID() : SnmpObjectId
+ getValue() : SnmpSyntax
+ setValue(value : SnmpSyntax) : void
+ getRequest(req : Request, index : int) : void
+ getNextRequest(req : Request, index : int) : void
+ setRequest(req : Request, index : int) : void
MIBTable
# oid : SnmpObjectId
# rows : SortedSet
# columns : MIBTableRow
+ getOID() : SnmpObjectId
+ addColumn(column : MIBLeaf) : void
+ addRow(index : SnmpObjectId) : void
+ getRequest(req : Request, index : int) : void
+ getNextRequest(req : Request, index : int) : void
+ setRequest(req : Request, index : int) : void
MIBTableRow
# index : SnmpObjectId
# items : SortedSet
+ getIndex() : SnmpObjectId
+ setIndex(index : SnmpObjectId) : void
+ addItem(item : MIBLeaf) : void
+ getItem(index : int) : MIBLeaf

Figure 5: *MIB Registry Class Hierarchy Diagram*

getAttribute or *setAttribute* on the JMX agent with the *mbeanName* and the *attribute* as parameters to access the attribute of the target MBean object.

A *MIBTable* class represents a SNMP table (tabular type managed object). A SNMP table may have multiple rows, and each row consists of multiple columnar objects. The *MIBTableRow* class is defined to represent a row of a SNMP table. It provides the methods to add MIBLeaf objects, which represent columnar objects in the row, and methods to get and remove them. The *MIBTable* object may contain multiple *MIBTableRow* object. The *MIBTable* contains a group of *MIBLeaf* objects named *meta columnar object*, which describe the structure information for the row. This group of MIBLeaf objects is organised as a *MIBTableRow* object named *columns*. When a new row is added, *MIBTable* will clone the *MIBTableRow* object to create a new *MIBTableRow* object. Each columnar object in the new row is the copy of the meta columnar object of its column, but with a different value. The *MIBTable* class provides the methods to manipulate columnar objects.

The *MIBTableProxy* class is a sub class of the *MIBTable* class. It acts as the proxy for a special kind of MBean object, which has a *TabularData* type attribute. *TabularData* is defined in the JMX specification and describes a table structure with an arbitrary number of rows that can be indexed by any number of columns (JMX1.2 2002). Each row is a *CompositeData* object, which is a hash map with multiple data items. The *CompositeType* object is used to describe the *CompositeData* object. All rows in a *TabularData* object must be associated with the same *CompositeType* object. This special kind of MBean object is automatically generated from the SNMP table by the MIB compiler (see Section 4.3).

The *MIBTableProxy* class supports a cache mechanism for efficiency. When a *MIBTableProxy* object initialised, *MIBTableProxy* queries the MBean (identified by the *MIBTableProxy*'s two attributes: *mbeanName* and *MBeanInfo*), and stores the MBean object's *TabularData* type attribute in the cache. The *MIBTableProxy* object also registers a notification listener for the MBean object. When the MBean's *TabularData* object is changed, the *MIBTableProxy* object will receive a notification and will query the MBean to update the cache. The cache mechanism is more efficient because the *MIBTableProxy* object does not need to contact the MBean when a SNMP management application performs *GET* or *GETBULK* actions on it. Only *SET* actions cause the *MIBTableProxy* object to update the MBean's *TabularData* object.

4.2.1 Generating Dynamic MBean Proxies

Both the *MIBLeafProxy* and *MIBTableProxy* objects act as proxies for a MBean. The SNMP operations performed on them are mapped to the accessor methods on the appropriate MBean objects, and then are forwarded to the MBean server. The MBean server finds the target MBean object, invokes the method on it and then returns the result or raises the exception. Proxies are dynamically generated from XML mapping files and are added into the MIB Registry. The mapping files define the mapping relationship between the MIB and MBean objects. Figure 7 provides an example of the mapping of a MBean object into the MIB.

The *RMIConnectorServer* is implemented as a MBean so that it can also be managed through protocol adaptors or connectors. There must be a relationship between the *RMIConnectorServer* and the nodes in the MIB; otherwise the SNMP adaptor has no idea how to map a SNMP request to the operations on the *RMIConnectorServer*. The *RMIConnectorServer* exposes four methods defined in the interface *JMXConnectorServerBean*, see Figure 7. Two of them, *isActive* and *getAddress*, are the get methods of the attributes *active* and *address*, and other two are operations according to design pattern described in the JMX 1.2 specification. Our prototype only supports the mapping of MBean attributes as SNMP does not support objects. The file *MBeansToMIB.xml* is used to describe how to map MBeans into the MIB. The mapping file assigns the OID "1.3.6.1.4.9876.1.1" to the MBean *RMIConnectorServer*, and describes the MBean's ObjectName so that the SNMP adaptor can locate the *RMIConnectorServer* instance through the MBean server. It also assigns the OID respectively to the attributes *active* and *address*.

The mapping file also maps the Java data type of the MBean attributes to the MIB data type. In this case, the SNMP adaptor loads the mapping file, and generates a *MIBGroup* object with two *MIBLeafProxy* objects which respectively represent the attributes *address* and *active*.

4.3 MIB Compiler

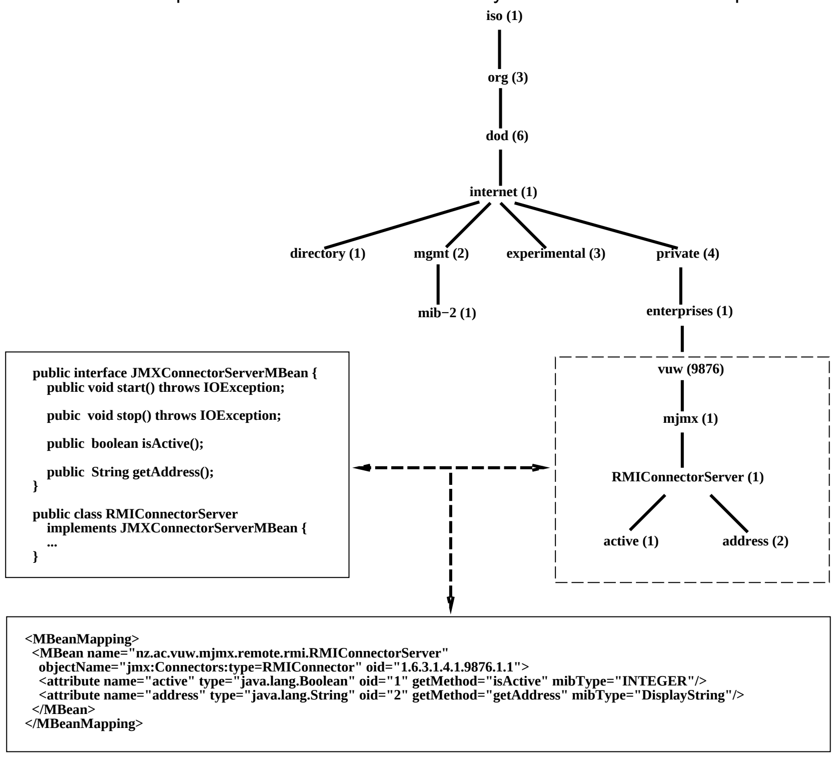
As described in Section 4.2.1, existing MBeans can be mapped into the MIB using the *MBeansToMIB.xml* file. However, JMX manageable resources must follow the design patterns and interfaces defined in the JMX 1.2 specification. Any incompatible resources must be instrumented as MBeans so that they can be managed by a JMX agent.

Our MIB compiler automatically generates MBeans representing a given SNMP MIB. The MIB compiler consist of two components: a MIB parser and a code generator. The MIB parser imports a MIB file and generates an intermediate representation. The code generator generates the Java source code and the XML file. The generated code is based on the JMX's model MBean specification (JMX1.2 2002) and can be used to create a model MBean on the fly. The generated XML is used to dynamically configure the model MBean. The model MBean provides management interfaces for non JMX compatible resources. This significantly reduces the programming burden and means that a developer can instrument existing resources according to the JMX specification as little as three or five lines of code.

4.3.1 MIB Parser

The MIB file is a normal text file written in Abstract Syntax Notation One (ASN.1) (ISO8824 1990) language, a formal language used to define abstract syntaxes of application data.

Rather than having single ASN.1 compiler with a lexer, a parser and a code generator, we utilise delegating compiler objects (DCO) (Bosch 1996), a novel approach to compiler construction that provides modular and extensible implementation of compilers. In DCO compilation is achieved through the cooperation of a group of compiler objects. A compiler object is only responsible for a particular part of the syntax, and has its own lexer and parser. The programming language is decomposed into a set of structure. Each structure is compiled by its associated compiler object. As shown in Figure 8, a MIB file can be decomposed into ten modules. The *TypeAssignment* module is used to define a new data type. The new data type can be Simple Type, Structured Type or Subtype. The *ValueAssignment* module is used to assign a value to a variable. In the MIB file, the *ValueAssignment* module is mostly used to assign a value to the Object Identifier variable. The *Import* module is used to import the types and variable



```
public interface JMXConnectorServerMBean {
    public void start() throws IOException;

    pubic  void stop() throws IOException;

    public  boolean isActive();

    public  String getAddress();
}

public class RMIConnectorServer
    implements JMXConnectorServerMBean {
    ...
}
```

```
<MBeanMapping>
  <MBean name="nz.ac.vuw.mjmx.remote.rmi.RMIConnectorServer"
   objectName="jmx:Connectors:type=RMIConnector" oid="1.6.3.1.4.1.9876.1.1">
   <attribute name="active" type="java.lang.Boolean" oid="1" getMethod="isActive" mibType="INTEGER"/>
   <attribute name="address" type="java.lang.String" oid="2" getMethod="getAddress" mibType="DisplayString"/>
  </MBean>
</MBeanMapping>
```

Figure 7: *Mapping a MBean into the MIB*

declared by other MIB files. The other seven modules, including *ModularIdentity*, *ObjectType*, *TextualConvention*, *ObjectGroup*, *NotificationType*, *NotificationGroup* and *ModuleComppliance*, represent seven macros defined in the MIB specifications. Our MIB compiler utilises a separate compiler object for each of the ten MIB modules. Each compiler object has a its own lexer and parser.

MIB compilation firstly eliminates all unneeded information in the MIB file (such as comments) and then passes the stream through the ten DCO compiler objects. Each DCO compiler performs its lexical analysis and then its parser provides the syntactic analysis. The entire syntactic analysis of the MIB file is the result of the collaboration of the different DCO parsers. Since only a very small subset of ASN.1 syntax is used in each DCO, the complexity of the implementation of DCO parser is significantly reduced. The output of DCO parsers are module objects representing the different modules. Module objects are divided into two groups: type groups and variable groups. Objects in the type group represent a data type, and objects in variable group represent an instance with type and value. Then, the module objects go through semantic analysis to check if objects are legal and meaningful (for instance, the values are valid, the types are defined, the compulsory attributes are set, and so on). The final step is to organise the objects as a MIB tree in terms of the OID value of each object. An optional XML file is also generated to represent the MIB file in the XML format.

4.3.2 Code Generator

The code generator walks through the MIB tree exported by the MIB parser and generates the instrumentation code and configuration files for the variables in the MIB tree. The code generator generates a Java class for each MIB group node. Every leaf node in the MIB group is represented by an attribute of the Java class. The corresponding accessor methods, such as *getX* or *setX* are defined in the Java class. In this case, *X* denotes the attribute name. Also, the code generator generates the Java class for each SNMP table. The methods to access the rows in the SNMP table are defined in this Java class. In addition, each generated Java class is associated with a XML configuration file that describes the mapping relationship between the Java class and the MIB. However, the generated instrumentation code only define the interfaces and provide skeleton code to describe how JMX incompatible resources can be accessed. The remaining manual tasks are to complete the skeleton code and implement the defined interfaces.

The generated Java classes can not be accessed directly by the JMX agent and must be wrapped into the model MBeans. The model MBean provides a set of interfaces which allow the JMX agent to perform the management operations on the resources wrapped in the model MBean object. The wrapping process starts by extracting the information for attributes, operations and notifications from the XML file associated with each Java class and then added this information to the model MBean. The whole process is done in one line code as follow. The method convertXmlToMBeanInfo converts the xml file into the MBeanInfo object which describes custom attributes, operations and notifications information, and then the RequiredModelMBean constructor use these information to construct a customised model MBean instance. The wrapping process is done automatically. The users can edit the file *JMXModelMBeanInfo.xml* to add the configuration file location and name for resources they want to wrap. When a JMX agent is initialzied, it checks out the file *JMXModelMBeanInfo.xml* and then generates the model MBean for the resources.

```
new RequiredModelMBean(convertXmlToMBeanInfo(xml));
```

4.3.3 A Code Generation Example

A code generation example for a MIB group is shown in Figure 9. The *system* group describes a set of objects common to all managed systems. It consists

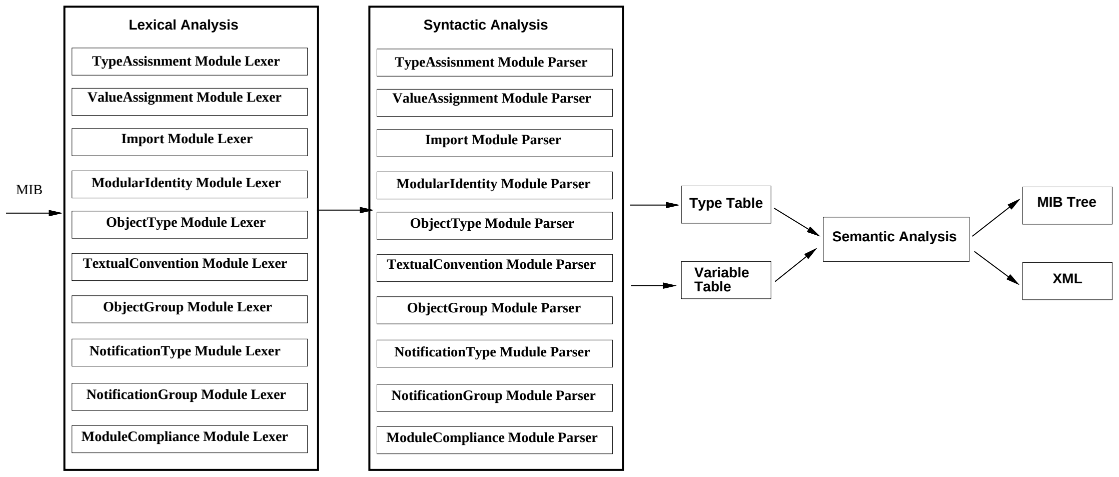

Lexical Analysis
TypeAssisnment Module Lexer
ValueAssignment Module Lexer
Import Module Lexer
ModularIdentity Module Lexer
ObjectType Module Lexer
TextualConvention Module Lexer
ObjectGroup Module Lexer
NotificationType Mudule Lexer
NotificationGroup Module Lexer
ModuleCompliance Module Lexer
Syntactic Analysis
TypeAssisnment Module Parser
ValueAssignment Module Parser
Import Module Parser
ModularIdentity Module Parser
ObjectType Module Parser
TextualConvention Module Parser
ObjectGroup Module Parser
NotificationType Mudule Parser
NotificationGroup Module Parser
ModuleCompliance Module Parser
MIB
Type Table
Variable Table
Semantic Analysis
MIB Tree
XML

Figure 8: *MIB Parser*

of eight scalar objects and a table object with four columns. The MIB compiler compiles the *system* group MIB file and generates five files:

- System.java: a Java class representing the whole *system* group except the table object *sysORTable*
- System.xml: a configuration file describing the mapping relationship between the *System* class and the *system* group in the MIB file
- SystemORTable.java: a Java class representing the table object *sysORTable*in the MIB file
- SystemOREntry.java: a Java class representing four column objects in the table object *sysORTable*
- SystemORTable.xml: a configuration file describing the mapping relationship between the *sysORTable* class and the *sysORTable* object in the MIB file

The *System* class in Figure 9 contains eight attributes that represent the eight scalar objects in the *system* group. The accessor methods for these attributes are also included. The *System* class does not include any OID information, but the configuration file *System.xml* describes the mapping relation between the attributes of the *System* class and the scalar objects of the *system* group. When the *System* class is wrapped into a model MBean and is registered within a JMX agent, a proxy object is dynamically generated from the *System.xml* file and is registered within the *MIBRegistry* in the SNMP adaptor (see Section 4.2.1). This proxy object will call the *System* class's *get* and *set* method upon *Get* and *SET* SNMP request.

The *SysORTable* class contains the attribute *sysORTable* which represents the table object of the *system* group. The row of the *sysORTable* is represented by the *SysOREntry* class. The *SysOREntry* class contains four attributes that represent four column objects. The access methods for these attributes are also defined. The *SysORTable* class defines the methods to manipulate the table object. The *getSysORTable* method is used to retrieve all rows in the table. The *updateEntry* method is used to update an existing row or add a new row in the table, and the *deleteEntry* is used to delete a row from the table. Similar to the *System* class, a proxy object is also generated from the *SysORTable.xml* and is associated with the *SysORTable* class.

4.4 SNMP Proxy

The SNMP adaptor makes JMX resources accessible to legacy SNMP managers. However, non-SNMP management applications can not access SNMP resources directly since they do not support the SNMP protocol. We have designed and developed a SNMP proxy to address this interoperability issue. As shown in Figure 10, the MIB supported by a remote SNMP agent is represented by multiple model MBeans. These model MBeans are registered within the MBean server and can be accessed by multiple protocols, such as Java RMI. These model MBeans act as proxies and the operations on them are forwarded to the appropriate remote SNMP agents through the SNMP proxy.

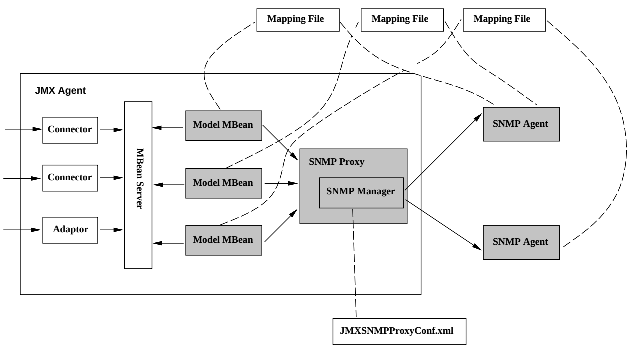
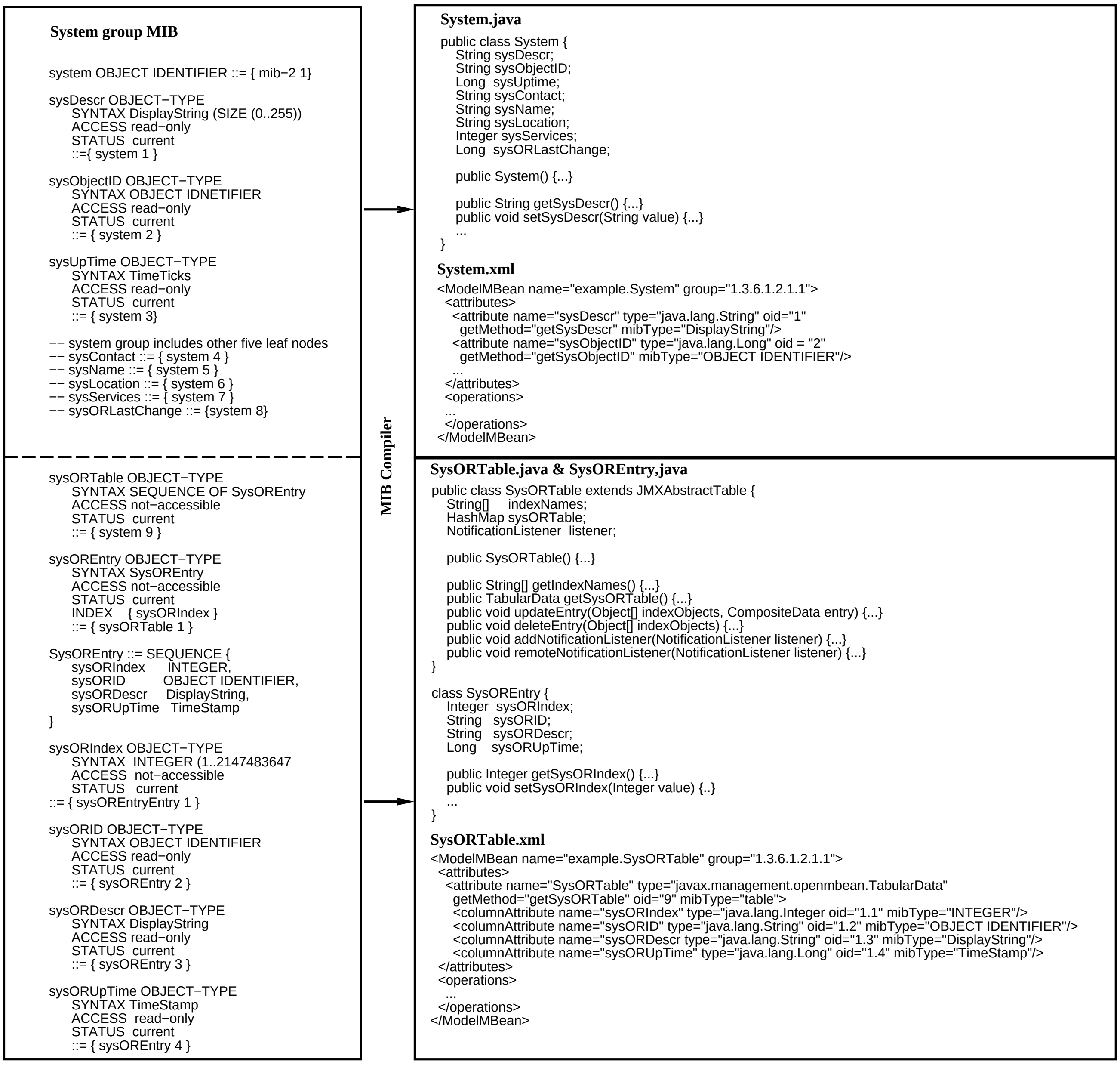


Figure 10: *SNMP Proxy*

To create proxy objects representing the remote SNMP agent's MIB, the MIB compiler is used to generated instrumentation code and xml configuration files from the remote SNMP agent's MIB file. However, only xml configuration files are used to create proxy objects. The instrumentation code is simply discarded. The JMX implementation used in this project provides two basic model MBeans: *RequiredModelMBean* and *JMXSNMPProxyModelMBean*. The *RequiredModelMBean* is used to instrument MBean incompatible managed resources. The operations on the *RequiredModelMBean* are forwarded to the managed resource. The *JMXSNMPProxyModelMBean* does not instrument any managed resources, but forwards the operations on it to the SNMP proxy. Generating a *JMXSNMPProxyModelMBean* instance using the toolkit is a single line of code (as shown below).



System group MIB

```
system OBJECT IDENTIFIER ::= { mib−2 1}

sysDescr OBJECT−TYPE
    SYNTAX DisplayString (SIZE (0..255))
    ACCESS read−only
    STATUS  current
    ::={ system 1 }

sysObjectID OBJECT−TYPE
    SYNTAX OBJECT IDNETIFIER
    ACCESS read−only
    STATUS  current
    ::= { system 2 }

sysUpTime OBJECT−TYPE
    SYNTAX TimeTicks
    ACCESS read−only
    STATUS  current
    ::= { system 3}

−− system group includes other five leaf nodes
−− sysContact ::= { system 4 }
−− sysName ::= { system 5 }
−− sysLocation ::= { system 6 }
−− sysServices ::= { system 7 }
−− sysORLastChange ::= {system 8}
```

```
sysORTable OBJECT−TYPE
    SYNTAX SEQUENCE OF SysOREntry
    ACCESS not−accessible
    STATUS  current
    ::= { system 9 }

sysOREntry OBJECT−TYPE
    SYNTAX SysOREntry
    ACCESS not−accessible
    STATUS  current
    INDEX    { sysORIndex }
    ::= { sysORTable 1 }

SysOREntry ::= SEQUENCE {
    sysORIndex      INTEGER,
    sysORID         OBJECT IDENTIFIER,
    sysORDescr      DisplayString,
    sysORUpTime     TimeStamp
}

sysORIndex OBJECT−TYPE
    SYNTAX  INTEGER (1..2147483647
    ACCESS  not−accessible
    STATUS   current
::= { sysOREntryEntry 1 }

sysORID OBJECT−TYPE
    SYNTAX OBJECT IDENTIFIER
    ACCESS read−only
    STATUS  current
    ::= { sysOREntry 2 }

sysORDescr OBJECT−TYPE
    SYNTAX DisplayString
    ACCESS read−only
    STATUS  current
    ::= { sysOREntry 3 }

sysORUpTime OBJECT−TYPE
    SYNTAX TimeStamp
    ACCESS  read−only
    STATUS  current
    ::= { sysOREntry 4 }
```

System.java

```
public class System {
    String sysDescr;
    String sysObjectID;
    Long  sysUptime;
    String sysContact;
    String sysName;
    String sysLocation;
    Integer sysServices;
    Long  sysORLastChange;

    public System() {...}

    public String getSysDescr() {...}
    public void setSysDescr(String value) {...}
    ...
}
```

System.xml

```
<ModelMBean name="example.System" group="1.3.6.1.2.1.1">
  <attributes>
    <attribute name="sysDescr" type="java.lang.String" oid="1"
      getMethod="getSysDescr" mibType="DisplayString"/>
    <attribute name="sysObjectID" type="java.lang.Long" oid = "2"
      getMethod="getSysObjectID" mibType="OBJECT IDENTIFIER"/>
    ...
  </attributes>
  <operations>
  ...
  </operations>
</ModelMBean>
```

SysORTable.java & SysOREntry,java

```
public class SysORTable extends JMXAbstractTable {
    String[]     indexNames;
    HashMap sysORTable;
    NotificationListener  listener;

    public SysORTable() {...}

    public String[] getIndexNames() {...}
    public TabularData getSysORTable() {...}
    public void updateEntry(Object[] indexObjects, CompositeData entry) {...}
    public void deleteEntry(Object[] indexObjects) {...}
    public void addNotificationListener(NotificationListener listener) {...}
    public void remoteNotificationListener(NotificationListener listener) {...}
}

class SysOREntry {
    Integer  sysORIndex;
    String   sysORID;
    String   sysORDescr;
    Long    sysORUpTime;

    public Integer getSysORIndex() {...}
    public void setSysORIndex(Integer value) {..}
    ...
}
```

SysORTable.xml

```
<ModelMBean name="example.SysORTable" group="1.3.6.1.2.1.1">
  <attributes>
    <attribute name="SysORTable" type="javax.management.openmbean.TabularData"
      getMethod="getSysORTable" oid="9" mibType="table">
      <columnAttribute name="sysORIndex" type="java.lang.Integer oid="1.1" mibType="INTEGER"/>
      <columnAttribute name="sysORID" type="java.lang.String" oid="1.2" mibType="OBJECT IDENTIFIER"/>
      <columnAttribute name="sysORDescr type="java.lang.String" oid="1.3" mibType="DisplayString"/>
      <columnAttribute name="sysORUpTime" type="java.lang.Long" oid="1.4" mibType="TimeStamp"/>
  </attributes>
  <operations>
    ...
  </operations>
</ModelMBean>
```

Figure 9: *A Code Generation Example*

```
new JMXSNMPProxyModelMBean(objectName, SNMPProxyRef,
                           convertXmlToMBeanInfo(xml));
```

The *objectName* represents the name of the model MBean object and the *SNMPProxyRef* is a reference to the SNMP Proxy. The convertXmlToMBeanInfo method converts the xml file into the MBeanInfo object that describes custom attributes, operations and notification information.

When a method of the *JMXSNMPProxyModelMBean* object is invoked, the *JMXSNMPProxyModelMBean* object forwards the invocation and object name to the SNMP proxy. The SNMP proxy checks the file *JMXSNMPProxyConf.xml* that describes the relationship between *JMXSNMPProxyModelMBean* objects and remote SNMP agents. For instance, an *JMXSNMPProxyModelMBean* object with object name "jmx:snmpagent:type=system" represents the System group of the MIB supported by a SNMP agent. This SNMP agent resides in the host "130.195.106.3" and listens on the port "161". After locating the SNMP agent, the SNMP proxy converts the invocation to a SNMP PDU, and sends it to the target SNMP agent.

5 Related Work

There are some commercial toolkits that provide broadly similar functionality to the work presented in this paper, such as Sun's JDMK toolkit (JDMK 1999). However, these are proprietary designs and their details are not available in the public domain. None-the-less, there are a number of important differences that we have been able to identify. For example, ordinary MBeans (those not generated by the JDMK's MIB compiler) can't be accessed by SNMP managers, whereas our toolkit enables ordinary MBeans to be accessible to SNMP managers via the "MBean-To-MIB" configuration file. Another difference is that JDMK generated MBeans are directly bound to the SNMP adaptor, whereas our MBean proxies are generated and bound at run time via the MBean Server. This is a cleaner more flexible solution, and conforms to the hourglass protocol model (Shanmugam et al. 2002).

6 Conclusions

The growing number of applications and services implemented in Java has increased the demand for Java based network management solutions. JMX provides a standard way to enable manageability for any Java based application, service or device (JMX1.2 2002). However, most existing management systems can not be managed directly via JMX compliant implementations. In this paper we present a toolkit that allows the rapid development of JMX agents, and that can interoperate with legacy SNMP-based network management systems. The core of this toolkit is a generic SNMP adaptor to enable JMX compliant agents to be accessed by SNMP-based management applications. A set of SNMP APIs have been developed to support the development of the SNMP adaptor. Several other tools have been developed to support the SNMP adaptor, these include: a MIB compiler that automatically generates MBeans representing a given SNMP MIB; and a SNMP proxy service to allow non-SNMP management applications to access the SNMP agent using a variety of protocols. A simple example is also given to illustrate the MBean generation process for a given SNMP MIB.

References

A. Puliafito & O. Tomarchio(2000), Using Mobile Agents to implement flexible Network Management strategies, Computer Communication Journal, 23(8):708–719.

Danny B. Lange & Mitsuru Oshima (1999), Seven Good Reasons for Mobile Agents, Communication of ACM, volume 42.

Frank Fock (2000), Agent++, An Object Oriented Application Programmers Interface for Development of SNMP Agents Using C++ and SNMP++, http://www.agentpp.com.

German Goldszmidt(1993), On Distributed System Management, In Processings of the Third IBM/CAS Conference.

IETF DIMAN Working Group (1999), Distributed Management (DISMAN) Charter, http://www.ietf.org/html.charters/disman-charter.html.

International Organization for Standardization (1990), Information Technology- Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).

International Organization for Standardization (1993), ISO 10165-1: Information Processing System - Open Systems Interconnection - Structure of Management Information - Part1:Management Inforamation Model.

Jan Bosch (1996), Delegating Compiler Objects: Modularity and Reusability in Language Engineering, Nordic Journal of Computing.

J. Schonwalder(1997), Network Management by Delegation - From Research Prototypes Towards Standards, In Processings of 8th Joint European Networking Conference.

K. Meyer, M. Erlinger, J. Betser, C. Sunshine, G. Goldszmidt & Y. Yemini(1995), Decentralising Control and Intelligence in Network Management, In Processings of International Symposium on Integrated Network Management.

OpenNMS (2002), joeSNMP API, http://sourceforge.net/projects/joesnmp/.

Paulo Simoes (1999), Enable Mobile Agent Technology For Legacy Network Management Frameworks, Technical Report, University of Coimbra.

Sun Microsystem Inc. (2002), Java Management Extensions Instrumentation and Agent Specification, v1.2

Sun Microsystem Inc. (1999), Java Dynamic Management Kit, http://java.sun.com/products/jdmk/.

Sun Microsystem Inc. (1999), JavaBeans Specification 1.0.1, http://java.sun.com/products/jdmk/.

R. Shanmugam, R. Padmini, S. Nivedita. (2002), Special Edition: Using TCP/IP, 2nd Edition, Que.

Y. Yemini, G. Goldszmidt & Mitsuru Oshima (1991), Network Management by Delegation, International Symposium on Integrated Network Management.

A Framework for Visual Data Mining of Structures

Hans-Jörg Schulz **Thomas Nocke** **Heidrun Schumann**

Department of Computer Science,
University of Rostock,
18051 Rostock, Germany,
Email: {hjschulz, nocke, schumann}@informatik.uni-rostock.de

Abstract

Visual data mining has been established to effectively analyze large, complex numerical data sets. Especially, the extraction and visualization of inherent structures such as hierarchies and networks has made a significant leap forward. However, it is still a challenging task for users to explore explicitly given large structures. In this paper, we approach this task by tightly coupling visualization and graph-theoretical methods. Therefore, we investigate if and how visualization can benefit from common graph-theoretical methods – mainly developed for the investigation of social networks – and vice versa. To accomplish this close integration, we introduce a design of a general framework for visual data mining of complex structures. Especially, this design includes an appropriate processing order of different mining and visualization algorithms and their mining results. Furthermore, we discuss some important implementation details of our framework to ensure fast structure processing. Finally, we examine the applicability of the framework for a large real-world data set.

1 Introduction

Visual data mining (VDM) has been proven to be an effective method to explore large data sets. It combines automated mining algorithms with visualization techniques. A variety of powerful methods and tools (e.g. the InfoVis (Fequete 2004) and the Prefuse (Heer, Card & Landay 2005) Toolkit and the systems Polaris (Stolte, Tang & Hanrahan 2002), Spotfire (Ahlberg 1996) or Visage (Roth, Lucas, Senn, Gomberg, Burks, Stroffolino, Kolojejchick & Dunmire 1996)) have been developed in the last few years. These tools combine linked views on the data with a high amount of interactivity, enabling users to switch quickly between automated and visual methods. Therefore, they integrate mining methods to explore numerical data from a variety of research areas, for instance AI, statistics and KDD. These methods can extract structures that are inherent in the data (e.g. hierarchical clustering). Furthermore, a number of visualization methods have been developed and integrated to visualize such abstract data as well as structures, gathered by the VDM process or already given with the data set. Examples for such structures are web link graphs and chemical molecule bonds. Another prominent example for such structures are social networks that apply methods to analyze social structures, i.e. to identify central nodes that can be understood as essential for the entire data set.

In this paper we want to discuss the question, if and how calculation methods from graph theory can be employed to essentially enrich VDM tools to explore structures. Our intention is to design a uniform framework that integrates a variety of well-known graph-theoretical and visualization methods for structures. For this purpose, dependencies of such methods have to be considered to design an appropriate control flow.

Up to now, the integration of graph-theoretical methods into VDM environments has not been in the research focus. A main reason for this is the high complexity of these algorithms that does not allow an interactive linking and brushing in VDM environments. To achieve this goal, special effort needs to be done.

Ankerst classifies current visual data mining approaches into three categories (Ankerst 2001). Methods in the first group apply visualization techniques independent of data mining algorithms. The second group uses visualization in order to represent patterns and results from mining algorithms graphically. The third category tightly integrates mining and visualization algorithms in such a way that intermediate steps of the mining algorithms can be visualized. In our approach we focus at the second level, separating the mining process into two parts:

1. execute time-consuming (automatic) algorithms and store their results, and then
2. enable users to do an interactive exploration of the structures in real-time, combining different visualization and less time-consuming graph-theoretical measures.

Although separating the time consuming execution of certain algorithms from the VDM process, performance issues are still of high relevance. Thus, efficient data structures and access mechanisms - managing both graphs and trees - are of high benefit for the interactive VDM (see e.g. (Fequete 2004)). In our framework implementation, we developed mechanisms that allows efficient storage of both structures and structural measures and algorithm results.

The paper is organized as follows: first we outline the background of graph-theoretical algorithms, visualization methods for structures and inspiring application areas (section 2). Afterwards, we discuss graph-theoretical methods suited for VDM, which especially includes their interaction with visualization techniques in section 3. Then, in section 4, we introduce our framework for VDM of structures. This includes the development of a general design and the discussion of internal data structures and their performance for different graph theoretical measures and algorithms. Afterwards, we discuss challenges of our approach and demonstrate its application to real-world data sets in section 5. Finally, we conclude the paper and discuss future work in section 6.

2 Background

On the one hand, in the last few years visualization of large structures, especially of trees and graphs, has been remarkably improved. Visualization techniques enable users to interactively explore complex structural relationships between the information objects. Well-known examples for hierarchy representations are (Lamping, Rao & Pirolli 1995), (Robertson, Mackinlay & Card 1991), (Shneiderman 1992), (Granitzer, Kienreich, Sabol, Andrews & Klieber 2004) and for networks examples are (Tollis, Eades & di Battista 1999) and (Brandes & Corman 2002).

A major challenge in this context is an intuitive navigation through large data sets to quickly find interesting patterns while preserving orientation. Therefore, focus+context techniques on structures have been developed (see e.g. (Gansner, Koren & North 2004), (van Ham & van Wijk 2004)).

A further challenge is the amount of data to be processed. Methods to explore and visualize huge structures that do not even fit in memory have been developed (e.g. (Abello, Finocchi & Korn 2001), (Abello & van Ham 2004)). Here, to ensure interactive data exploration, mechanisms that decide to precompute long-lasting algorithms needed to be developed.

On the other hand, there is a variety of automatic methods introduced by graph theory to explore structures. General work has been done to determine the complexity of graph-theoretical algorithms and to estimate their efficiency and effectivity for practical data sets (e.g. (Valiente 2002)). Furthermore, these methods have been applied and refined for practical application fields, such as social sciences (e.g. to detect community structures within social and biological networks (Girvan & Newman 2002)) and biotechnology (e.g. the usage of generalized interval graphs to solve the physical mapping problem that occurs when sequencing fragments of DNA (Zhang 1994)).

Moreover, there are a few approaches to apply graph-theoretical measures to parameterize visualization and vice versa. For instance, van Ham and van Wijk (van Ham & van Wijk 2004) use hierarchical clustering on graphs and represent nodes in the focus in another hierarchy level than nodes in the context. Other examples are (Abello & van Ham 2004), (Frischman & Tal 2004) and (Gansner et al. 2004).

However, a systematic approach that integrates measures and algorithms of graph theory with interactive visualization methods is still missing. In fact, this can be very helpful to support VDM of structures, for instance to select and parameterize tree and graph visualization by graph measures. Therefore, nodes of high connectivity or of other specific values of interest can be laid out into the focus. Moreover, a tree visualization technique resp. a graph visualization technique can be chosen to represent a structure in dependency of its similarity to a tree. If there are only a few edges to be deleted from a graph to form a tree, a tree visualization technique can be a good choice to visualize this graph, representing the non-tree edges in another way (see figure 1 right and (Fekete, Wang, Dang, Aris & Plaisant 2003)). If, on the other hand, the graph is less similar to a tree, tree visualization techniques are not appropriate (see figure 1 left), and a default graph-drawing technique is the better choice.

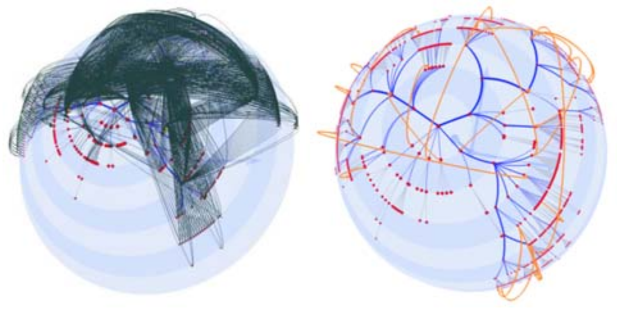


Figure 1: Networks represented by the tree visualization technique MagicEyeView (Kreuseler & Schumann 2002) with a large (left) and a small (right) number of non-tree edges [images taken from (Voigt 2001)].

3 Algorithmic mining techniques for complex structures

Even though the visualization is a powerful and important part within the concept of VDM, each visualization technique has its limits on how much data it can possibly display. This most critical problem occurs rather often when analyzing real-world data, but it can be overcome to some extent by appropriate information-hiding, brushing or focus+context techniques embedded within the visualization used. To parameterize those techniques, further information about the given data set can be computed by a thorough algorithmic pre-processing:

- irrelevant data, like statistical outliers that can be hidden
- somehow "important" data that needs to be emphasized
- bits of data that are very similar and can be clustered

The field of graph theory already provides a wide variety of such mining techniques (i.e. finding maximum cliques, shortest paths or calculating modular decompositions). Different domains, like the theory of social networks, the so-called *web structure mining* that is used by search engines throughout the WWW or the bio-chemical analysis of protein structures, supply further methods for analyzing large amounts of structured data. Roughly, these methods can be divided in three categories which can be subtle interrelated or simply used one by one if needed:

- structural measures that capture important attributes of the graph (the list in section 3.1 is based upon an overview given in (Brandes & Wagner 2003)),
- clustering algorithms to decompose large structures,
- methods for graph matching to identify and locate substructures of interest.

Additionally to these three categories, efficient approaches to automatically preselect well-fitting methods can help to maintain a comprehensive overview about the huge number of applicable graph algorithms (section 3.4).

3.1 Structural measures

Structural measures can be computed locally (separately for each node) or globally (for an entire graph or subgraph). Local measures of interest are i.e. the following centrality measures, which can be understood as an index of how "important" a node is (The examples after the formal description of each measure refer to figure 2):

- The **node degree**, that returns the number of incident edges for a node v (i.e. $\deg(c) = 3$).
- The size of the **k-neighborhood** $|N_k(v)|$, that equals the number of nodes within distance $\leq k$ from a given node v (i.e. $|N_1(c)| = 3, |N_2(c)| = 5, |N_3(c)| = 6$).
- The summed up lengths of all shortest paths from a node v to every other node yield the **closeness** of that node
(i.e. $\mathrm{cls}(c) = 1 + 1 + 1 + 2 + 2 + 3 = 10$).
- The maximum length of all shortest paths from a node v to every other node equals its **eccentricity**
(i.e. $\mathrm{ecc}(c) = \max(1, 1, 1, 2, 2, 3) = 3$).
- The **node betweenness centrality** of a node v, which is defined as the number of all shortest paths that pass through v (i.e. for c: 4 from a and b, 2 from d, e, f and g, that sums up to 16).

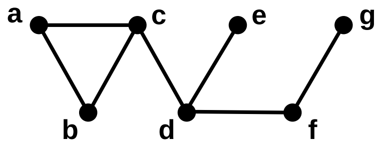


Figure 2: An example graph G(V,E).

The following similarity measures on the other hand can be used for graph clustering or within the layout algorithm to position comparable nodes closer together:

- The **connectivity** of the nodes u and v is the minimum number of edges that have to be removed from the graph in order to separate both nodes in a way that no path between them exists (i.e. $\mathrm{conn}(a, b) = 2$).
- The **dependency** of node u from node v returns the number of shortest paths originating in u and passing through v (i.e. $\mathrm{dep}(c, d) = 3$ and $\mathrm{dep}(d, c) = 2$).

Since different concepts of centrality and similarity stand behind those measures, their outcome often differs in many ways, as can be seen when comparing connectivity (a symmetric measure, since $\mathrm{conn}(u, v) = \mathrm{conn}(v, u)$) with dependency (usually asymmetric, except for some graph classes like circles or cliques). The user must be aware of these differences at all times and should choose the most suitable and expressive measure to model his special goal of analysis.

While these local measures are available only for single nodes, the more general global measures give an overall view of the structure. The simplest form of a global measure is of course the average of a local measures (i.e. the average node degree) that can easily be computed. Other global measures are:

- The **diameter** $diam(G)$ of a graph $G(V, E)$, which equals the largest eccentricity value, or the **radius**, which equals the smallest eccentricity value. (i.e. $\mathrm{diam}(G) = 4$, $\mathrm{rad}(G) = \mathrm{ecc}(d) = 2$)
- The **compactness** or **density** of a graph that provides information about how many of all possible edges are actually present. (i.e. G got 7 out of 21 possible edges)
- A **treelikeness-value** can be computed to obtain a measure for structural resemblance with a tree (a graph without any induced circles). Among many existent treelikeness measures, we introduce an adaption of this term that is optimized for the use within the visualization process: a graph is called (p, k)-treelike, if it has no more than k cross-edges and the fraction of cross-edges with respect to all edges is less than the percentage p.

Such global measures can be very useful for the determination of an appropriate visualization method: i.e. an underlying treelike structure with only a few crossedges can be identified as such by its treelikeness-value and hence laid out with a hybrid approach as described in section 2. This approach generalizes the ideas introduced in (Fekete et al. 2003), where a tree visualization is extended in a similar way.

3.2 Graph clustering

Over the years different flavors of clustering have been developed and evolved further. The clustering techniques that are most often used for the purpose of VDM are the hierarchical clustering algorithms. They do not only yield a clustering of a desired granularity but also a way to explore the data set via browsing the clustering's dendrogram (Herman, Marshall & Melançon 2000). Hierarchical clusterings can be computed either bottom-up by combining similar elements (normalized-association-method (Shi & Malik 1997), single- or average-linkage-method,...) or top-down by separating elements that differ (normalized-cut-method, edge-betweenness-centrality-clustering,...). Hence for both approaches, similarity or distance measures need to be computed beforehand.

One way to circumvent this need is the use of graph decompositions, which also results in a hierarchical graph partition. They work directly on the graph's structure without any additional measures needed and can usually be computed in linear time. Examples are: modular-decomposition, k-core-decomposition (Batagelj, Mrvar & Zaveršnik 1999) or decomposition through distance-k cliques (Edachery, Sen & Brandenburg 1999).

3.3 Graph matching

The search for special structures within the data set is a tedious task that is difficult to automate. Several different kinds of graph matching can be used:

- The **exact graph matching** searches for a subgraph that is identical to a specified pattern (Subgraph Isomorphism Problem)
- The **inexact graph matching** searches for a subgraph that is as similar as possible to a specified pattern.
- The search for the largest given configuration, i.e. the largest clique or the longest path.
- The detection of the most frequent subgraph of given minimum size.

Since all of these matching problems are quite complex from an algorithmic point of view, mostly heuristic approaches are used to find approximate solutions (Bunke 2000).

3.4 Semi-automated technique selection through metadata

To achieve a certain mining goal, different algorithmic methods can be applied. Usually some of them fit a particular case better than others. To determine suitable techniques, metadata describing the overall structure can be used to derive helpful indications on which method to employ.

An example would be the choice of fitting runtime-efficient algorithms depending on the graph's overall structure. As already mentioned, many of the described graph theoretical methods are painfully slow due to their polynomial or even exponential runtime complexity. An algorithm is usually considered to be efficient on very large data sets, if its complexity is subquadratic. The lack of efficient algorithms results in intolerable high computation times and prevents interactive techniques (*time bottleneck*). But in case the data set fulfills certain structural constraints, efficient algorithms do exist for most of the above presented problems (graph matching, clustering, decomposition, etc.) Since it is widely known that academic worst-case-constructions occur rather seldom in real-world-scenarios, some of the desired constraints are virtually always fulfilled.

An example would be the *sparseness* of a graph, which means that the number of edges is much less than the possible number of edges within the graph. Most graphs from different areas are sparse, e.g.:

- A biochemical molecule must be sparse, since every atom can have only a small number of chemical bonds.
- A social network is usually sparse, because a person normally does not have some hundred acquaintances.
- A large website rarely links from each hypertext document to every other document, as well as scientific papers do not cite every other paper in their field and vice versa.

Therefore, algorithms can be optimized to take advantage of the sparseness and compute their results in less time. An example for such an optimized algorithm is the method to determine the node-betweenness-centrality as described in (Brandes 2001). Another example is the k-core-clustering (Batagelj et al. 1999) that decomposes the data set within a linear timebound with respect to the number of edges:

- In the worst case, the graph $G(V, E)$ features all of its possible edges $|E| = |\wp_2(V)| = \frac{1}{2} \cdot |V| \cdot (|V| - 1)$ and has therefore a quadratic runtime bound with respect to the number of nodes.
- In the average and more practical case, the graph is usually sparse and contains only a small fraction of its possible edges. So the runtime complexity will be subquadratic (in terms of the size of the node set) and thus acceptable.

Besides the already provided global structural measures like density or treelikeness (see section 3.1), other structural descriptors can be useful:

- Testing whether a directed graph is **acyclic** can lead to very efficient algorithmic solutions to many NP-complete graph problems that are hard to solve on arbitrary graphs. This test runs in linear time and tries to sort the data set topologically. If this succeeds, the resulting topological ordering can be used as input for fast algorithms that have been especially adapted for this case.
- Determining the **data relationship** (Bertin 1981) that gives an impression of how the overall structure is organized: linear, circular, hierarchical, etc.

These descriptors can also be used to select and parameterize an appropriate visualization technique that can be especially suited to display exactly the described kind of a structure. An example for this method would be the graphs shown in figure 1: contrary to the right part of figure 1, the treelikeness-value of the graph on the left side is obviously out of the range and the MagicEyeView-technique actually not applicable.

Additionally, certain graph classes can even be visualized in very special manners. For instance, if a graph is detected to be an interval graph, it can be displayed as an intersection model consisting of intervals on a straight line. To view a graph as such an intersection model is quite common in genetic engineering and computational biology. Furthermore many algorithmic problems can be efficiently solved on interval graphs. Hence a set of detection-procedures for certain graph classes of interest could further yield useful hints for the choice of a well-suited visualization and speed up the mining process if tailor-made implementations for the detected graph class are provided.

4 A general framework for Visual Data Mining in complex structures

4.1 Design criteria

Designing a visualization or VDM framework is a sensible process. Many decisions made in early development stages are complicated to redo. Furthermore, a variety of backgrounds with varying data sets and tasks require varying software architectures. In the following, we list five main design criteria for a VDM framework for structures:

- **Generality**
  - Adaptability to ***different application backgrounds*** (e.g. social networks, organic chemistry),
  - Scalability to ***various users*** with varying background knowledge,
  - ***Modular design*** allowing to plug in any kind of visualization techniques and mining operators on structures
- **Flexibility**
  - Flexible ***control mechanisms*** to select, connect and parameterize measures, mining algorithms and visualization techniques on structures (e.g. using scripts, or interactively using menus or data flow charts),
  - ***Visual queries*** with a direct visual feedback,
  - ***Support to derive additional data*** to gain a deeper insight into data features (e.g. by extracting relevant substructures)
- **Usability**
  - ***Data abstraction*** to get easy access to different kinds of structure data sources independent of their internal and external storage format,
  - ***Acceptable reply times*** of calculations (approximation techniques might need to be considered in case of unfavorable runtime complexities or a low upper time bound given by the user),

  - ***Intuitive means of interacting*** with even complex mining methods

- **Efficiency**
  handle fairly large data sets and avoid screen, storage space and temporal bottlenecks
  - ***Memory Efficiency*** through smart data structures as described in 4.3.
  - ***Runtime Efficiency*** by decoupling the interactive parts of the VDM-process from the non-interactive ones as discussed in 4.2.
  - ***Screen Efficiency*** to effectively apply the whole screen space displaying large structures

- **Task orientation**
  Can the user fulfill all tasks to gain the exploration target? This includes a variety of paradigms, including the following:
  - Focus+context
  - Overview+detail
  - Brushing
  - History (especially Undo and Redo)
  - Sorting and filtering
  - Zooming.

However, it is not reasonable to design a VDM framework that is applicable for all kinds of possible applications and tasks. Thus, we design a general architecture for the VDM of structures that can be easily extended by any measures and methods. Therefore, in the following section, general modules in the field of structure exploration and their processing will be introduced in an abstract scheme. These modules are containers for measures, visual and non-visual mining methods as well as for units supporting general tasks such as dynamic queries or history.

4.2 Conceptual foundation

In the following, we introduce our design of a VDM-framework that consists of several different functional modules:

- interfaces for user interaction before and after the extensive mining operations,
- a preprocessing unit and a unit to compute structural descriptors,
- the algorithmic kernel that does the mining and lays out the data for its graphical representation.

Thus it is possible to extract the complex algorithmic kernel to do extensive calculations without the need for user-input on different, eventually faster machines, while the user interaction before and afterwards is done within the framework itself. This is the only way to efficiently process the needed graph theoretical algorithms, since one cannot work around the fundamentals of complexity theory. A schematic overview of the framework is depicted in figure 3, where the several fragments are colored according to their function within the overall VDM-process. For its modular design as required by the design criteria in section 4.1, the fragments can be extended with different visualization techniques and algorithmic modules, to realize their function in detail. The fragments provide the following functionality:

- During the **initial interaction**, the user specifies additional properties of the structure that are not explicitly included in the data set. Here the user should also be able to roughly parametrize upcoming algorithmic computations by setting upper runtime bounds and the like.

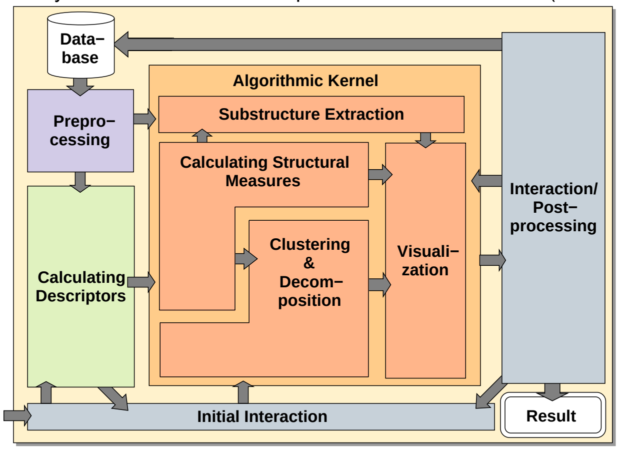


Figure 3: Our general VDM-framework design.

- The **preprocessing** can be used for cleansing and filtering the data.
- The **calculation of descriptors** tries to gather enough metadata as discussed in section 3.4 enabling the user to determine fitting mining and visualization techniques.
- The **algorithmic kernel** does the actual work of calculating additional data, which is one of the crucial features a VDM-framework must fulfill: computing measures (like those in section 3.1), extracting substructures, clustering or decomposing the graph (see section 3.2) and finally calculating a graphical layout for the resulting data.
- The **interaction** on the gained graphical representation is used for the actual visual exploration of the data set. Here some post-processing can be done to further manipulate and query the data set interactively through the visualization and to write back those changes to the data base.

For each of these computational steps, user defined modules can be plugged into the framework and executed in the given order. Usually, the VDM-process based on this architecture starts off by determining promising mining methods through analyzing the automatically computed descriptors and taking the user's goals of analysis into account. Afterwards, the chosen techniques will be employed upon the data and their results will be stored for further visual investigation later on. Depending on how the results are structured, an appropriate visualization technique is selected and used to generate an interactive overview of the outcome that can be graphically explored. As demanded by the design criteria, a wide variety of navigational elements, filtering and searching techniques can be provided through a standardized user interface that applies to all modules. Figure 4 shows a detailed view on the framework with several representative modules plugged-in to illustrate typical operations within the different fragments. In detail, the modules shown in figure 4 add the following functionalities to the framework:

Modules for the initial interaction:
After starting-up the framework, these modules provide a first possibility to augment the mining process with additional information about the data, the mining goals and any given constraint on the mining process. For example by distinguishing between *undirected and directed graphs*, the user

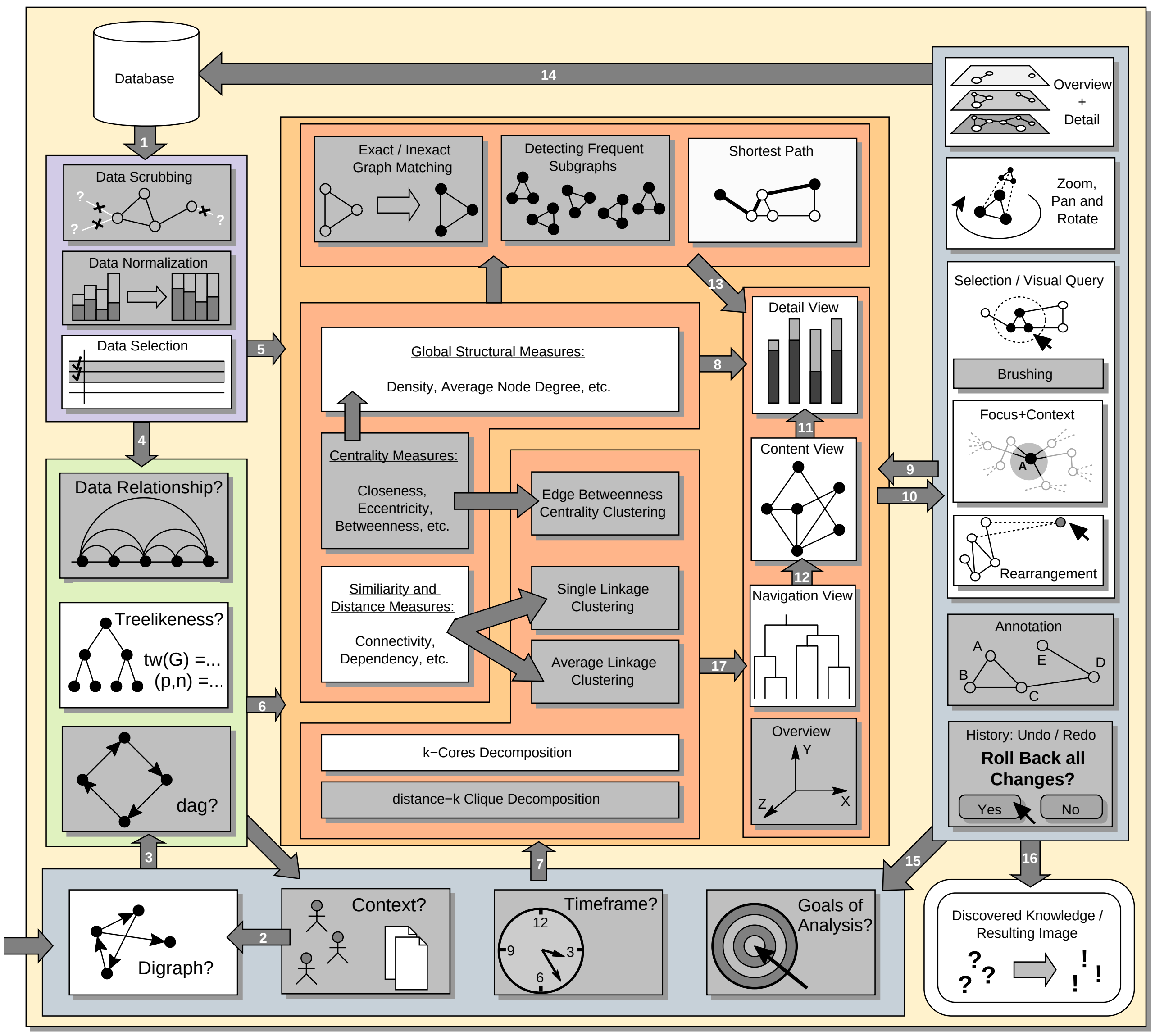


Figure 4: Our VDM-framework design with representative modules. Greyed out Modules have not yet been implemented or their implementation has not yet been adapted to be used within the framework.

indirectly influences the choice of applicable algorithms. Sometimes, this can also be deduced directly from the *context of the data set* – i.e. link structures in hypertext documents always form a digraph. Additional information that affects the selection of appropriate algorithms can contain *upper runtime limits* to insure acceptable reply times or explicitly stated *goals of analysis* (Nocke & Schumann 2004). These modules directly fulfill the design criteria for adaptability to different application backgrounds and scalability to users with varying background knowledge.

Modules for the preprocessing:
The modules presented in this fragment are responsible for data *cleansing* from measuring errors such as dangling ends in the set of edges. Furthermore, in the preprocessing phase a *data selection* and edge weight *normalization* can be performed etc. Transformations like these allow to define a standardized, abstract data format to work with, which is an essential design criteria.

Modules for the processing of structural measures:
In case of a digraph structure, this fragment could test a graph for *acyclicity*, which might again trigger additional goals of analysis and a selection of especially optimized algorithms in the algorithmic kernel. Furthermore all of the discussed measures in section 3.1 can either be calculated here, or at least be approximated if the exact calculation would be too time consuming.

Modules for the algorithmic kernel:
The functionalities of most of the modules shown in this fragment have already been addressed in section 3. Appropriate graph-algorithmic and visualization modules are selected via the calculated measures as described in 3.4. Since there usually does not exist a single optimal graphical representation of every aspect of the data, we propose a fourfold approach in the style of the ideas on multiple views in (Scharl 2002). The data set's inherent structures that have been computed beforehand are presented in a *navigational view*, that makes them accessible in a hierarchical manner. Once a region of interest has been selected in the navigational view, the sub structure associated with it will be shown in a *content view*. Selecting a node, an edge or a substructure within the content view will again trigger a detailed graphical representation of those in the *detail view*. To prevent loosing the orientation within the data set while visually exploring it, a rather static *overview*

should be provided to aid at keeping the global mental map of the structure. It is further possible to use multiple instances of each view that utilize different visualization techniques. This view-concept ensures a simple and uniform way to explore the data while using the available screen space most effectively.

Modules for the interaction and postprocessing:
This fragment contains the usual interactions and manipulations upon the visualized data set, which enable the user to carry out common exploration tasks as stated by the design criteria. This includes interactive annotation to add supplementary comments to the presented clusters and substructures and a similar history concept to the one we have developed and presented in (Kreuseler, Nocke & Schumann 2004).

4.3 Implementation

The functional modules introduced in figure 4 have been implemented in an interactive framework that is based on C++ and the Qt-Library. White colored modules are fully-integrated, and grey modules are under development.

A major challenge developing this framework was to implement an efficient storage concept for structures. In this context, a lot of different approaches to store graphs have been discussed in literature. Besides delegating storage issues to 3rd party products like relational databases, in practice only three techniques are widely used: **adjacency matrices** for fairly small graphs, **object-based data structures** for medium-sized graphs, that use objects for each node and link them via pointers, and finally **table-based structures** that contain the nodes and edges along with their attributes in a simple linear order, which is suited for large graphs. For its small memory overhead, most frameworks for processing large graphs utilize the last approach. A popular example is the Java-based implementation of such a linear storage for large graphs by the InfoVis ToolKit (Fequete 2004) which extends the table-based approach with several additional features, like fields for **auxiliary metadata** on each column and the ability of **masking** several nodes or edges within the list by toggling certain selection-bits. Beside these, our table-based data structure adds the following functionality:

- Columns may contain **permutations** of the table's rows, i.e. a topological ordering or the sequence of a breadth-first-search. That allows to store multiple orderings and eradicates the need to shuffle around the table's rows to sort them.
- Besides ordinary values, a cell of the table may contain an entire list to efficiently store **adjacency lists** or even **hypergraph structures** with a variable number of nodes per hyperedge within the same data structure.
- Each column can exist as a **placeholder** only, which will be filled in automatically when it is used for the first time. This pushes file-reading operations and computations as far back as possible and may save time and memory footprint in an average use case.

To improve the speed of look-ups in huge lists, a simple caching layer is used which allows direct access to the last couple of entries that where used. This storage concept addresses mainly large and complex structures that take up a lot of space in memory – up to the point, where they just will not fit in there anymore (*memory bottleneck*). It counters this case through a layered set of predefined behaviors, from which can be chosen automatically or interactively:

1. To push storage problems as far as possible, a strong emphasize of the framework lies upon an efficient storage of the data, that is painstaking space-saving and still tries to maintain an acceptable average access time. This is done by using the before mentioned table-based approach that can be augmented with supplementary data if enough memory is available or if an algorithm definitely needs it. Furthermore, the data set can be split up in smaller subsets if the overall structure has unconnected components that can be computed separately.
2. In case the data still does not fit into the memory, unneeded attributes like edge weights or previously computed measures that are not vital to run a specific algorithm, will not be loaded into the memory unless the user explicitly says so (*placeholder columns*).
3. If the memory's sufficiency is of further concern, all standard modules of the framework must be exchangeable with external algorithms that are especially optimized for this special case.
4. For those modules that do not provide a special external version, a smart caching layer has to be introduced to the frameworks I/O to minimize memory swaps.

In most cases, the first layer that employs a deliberate usage of memory will be effective enough to fit all data in. For larger data sets, the framework has to utilize one of the latter layers until every needed attribute can be accessed. Moreover, it is imaginable to introduce additional intermediate layers like certain garbage collection functionalities or semi-external versions of frequently used modules that make use of the situation where at least the node set fits in the main memory. This would further increase the chance to prevent the use of generally slower external algorithms. Thus far, the first two of the presented layers have been successfully implemented.

4.4 Graphical user interface

Sometimes, when trying to visualize a set of data, it turns out that the amount of data is just too large to fit the output device, i.e. the data objects that should be displayed outnumber the available pixels (*screen bottleneck*). To reduce the visual complexity in order to circumvent this bottleneck, additional time consuming clustering steps might thus be needed. For the sole purpose of decreasing the structural complexity through node-aggregation, we propose the use of heuristics or graph decompositions that have a linear runtime bound. Ideally this reduction produces a hierarchy that can be used to filter the results interactively up to a desired level of detail, like the mentioned k-core-clustering. Figure 5 presents yet another method to explore a potentially overloaded and overdetailed graphical representation. Since there is no visualization method, that serves all demands equally well, different views upon the data are generated as needed and functionally linked as described in section 4.2: The **content view** is shown after the initial layout is calculated by an appropriate visualization module (in dependency of the treelikeness-value we use Fruchterman-Reingold's spring embedder method (Fruchterman & Reingold 1991) for graphs and the MagicEyeView for trees and treelike graphs) and provides several possibilities of interaction: zoom, rotation, selection, filtering, etc. The output can be interactively constrained to only those nodes that have a certain centrality measure within the range defined by the sliders at the right. An additional **navigation view** is based upon the dendrogram of a hierarchical clustering. We used a subsequent display of MagicEyeViews (Kreuseler & Schumann 2002) to visualize the huge hierarchical structure of the dendrogram, where each selected cluster can itself be a root-node within a newly generated subview. As an example for a **detailed view**, figure 5 shows the distribution of the k-neighborhood from section 3.1 for $k = 1 \dots 5$. By selecting a certain neighborhood through this display, the content view can be adapted to show it for exploratory analysis.

4.5 Summary

The proposed framework architecture has a modular, extensible design. It has a general underlying data structure, and thus, can handle various structures from different backgrounds. Explicitly integrating the application context and user goals makes it scalable to various users from different domains. It offers both automatic and interactive mechanisms to control measures and techniques in the algorithmic kernel, especially enabling users to specify queries visually (for instance to select substructures). It enables users to specify, derive and apply additional data (metadata) which can be used for the semi-automatic selection of suitable algorithms, lead users through the exploration process and increase the user knowledge about the handled structures. Furthermore, the architecture supports to handle large data sets by smart data structures and by a control mechanism for long-lasting resp. interactive processable calculations. Interactive visualization techniques have been integrated, even applicable for large data sets. Therefore, they apply focus+context, overview+detail and brushing+linking paradigms and support interactive sorting, filtering as well as navigational support.

5 Case study

We successfully applied our framework to a variety of data sets. This included a web link graph of our institute internet pages (51497 nodes, 425247 edges), a citation network (509 nodes, 1551 edges) and peer-to-peer-networks with a few hundred nodes. For this paper, we demonstrate the usefulness of our framework design and present interesting insights for the medium sized Edinburgh Associative Thesaurus data set (short EAT, see http://www.eat.rl.ac.uk). For the following exemplary analysis, we present the exploration process closely related to the flow of the design chart from figure 4, giving details about the depicted modules and the arrows interrelating them.

The EAT data set consists of an empirical set of word associations (Kiss, Armstrong, Milroy & Piper 1973). Therefore, a list of 8.210 very frequently used English words (stimuli) has been compiled and the associative responses from test persons were gathered. Since the responses themselves are not necessarily stimuli, we eliminated these dangling ends in a data scrubbing preprocess (fig. 4, arrow 1). The resulting graph contains of 8.210 nodes (the stimuli) connected via 261.453 weighted edges.

Then, in an *initial interaction* step, context knowledge about type and history of the data set leads to the specification of the graph as a digraph (arrow 2). Based on this knowledge, a variety of descriptors can be calculated for the preprocessed digraph (arrows 3 and 4). This includes to calculate the *treelikeliness* which is relevant for the later selection of an appropriate visualization (as described in section 3.1). The actual (p, k)-treelikeness of the given data set results to (3.1%, 253244), which means that 253244 edges would have to be deleted in order to convert the network into a tree — only 3.1% of all edges would remain to form the spanning tree.

Based on these calculations and specifications, we started the main exploration phase in the *algorithmic kernel* (arrows 5, 6 and 7). First, important global structural measures have been calculated. For instance, to estimate the graph connectivity, we calculated an *average node degree* of 31.85, which means that each stimulus-word is associated with approximately 32 other stimuli-words. Based on this medium *average node degree* and due to a low *treelikeness* value we concluded that the EAT graph is a medium connected network and not suited to be laid out with a tree visualization technique. Hence, we chose a network visualization as content view (arrow 8). Therefore, to get a first overview of 8.210 nodes (arrow 7 and 9), we actually used a 3D-spring-embedder network visualization technique (Fruchterman-Reingold (Fruchterman & Reingold 1991)). This computation lasted about an hour (on an Intel PentiumM 1.4GHz machine with 512 MByte RAM), and thus, was executed non-interactively within the algorithmic kernel.

Then, to get more details about certain nodes, the user can *zoom*, *pan* and *rotate* the graph layout, as well as *select* certain nodes (arrows 9 and 10). Furthermore, to filter the crowded representation (arrow 9), we calculated the associative neighborhoods of chosen words based on the structural measure *1-neighborhood*. Then, using the two sliders depicted on the right side of figure 5, the user can fade out all nodes that do not have a *1-neighborhood-size* within a certain range (arrows 8, 9 and 10). For instance, in figure 5 we applied a 1-neighborhood-filter of 165 as minimum and a value of 1106 neighbors as maximum. Thus, the user can investigate a sparsely crowded graph with the main associated stimuli, leaving out all stimuli that are lesser associated. The maximum value of 1106 belongs to the node of the word MAN, which is the most associated word in this data set. The next heavily associated words are GOOD and SEX (ca. 870 associative links to other words).

Further, the user can get details-on-demand about these selected nodes (fig. 4, arrows 9, 10 and 11), displaying the k-neighborhood-diagram of a selected node in a *Detail View* (arrows 8 and 9). This gave us another interesting insight: most nodes lie within a distance of 3 (see in the k-neighborhood-diagram in fig. 5). This is a further proof for the observed medium to high density of the graph and indicates that there are no isolated substructures.

Then, to get a grip at this highly interrelated graph, we clustered the graph hierarchically using a k-core-decomposition. This computation lasted a couple of minutes, and was also executed within the algorithmic kernel. Using the resulting dendrogram, the user can explore the whole graph in an *overview+detail* manner, *focusing* on certain hierarchy levels (fig. 4, arrows 8, 9 and 17). Therefore, in a *navigation view*, certain levels of the hierarchy are displayed in a *MagicEyeView* (see fig. 5), and the user can interactively focus on certain clusters and hierarchy leaves of interest, still keeping the context visible (fig. 4, arrows 9 and 10). Furthermore, clusters of interest can be selected in the *MagicEyeView*, to explore the subgraphs they induce within the *content view* (arrows 9, 10 and 12). A *brushing* mechanism displays these subgraphs. Examples for words that have automatically been clustered together are:

- ITS, MUST HAVE, POSSESSIVE

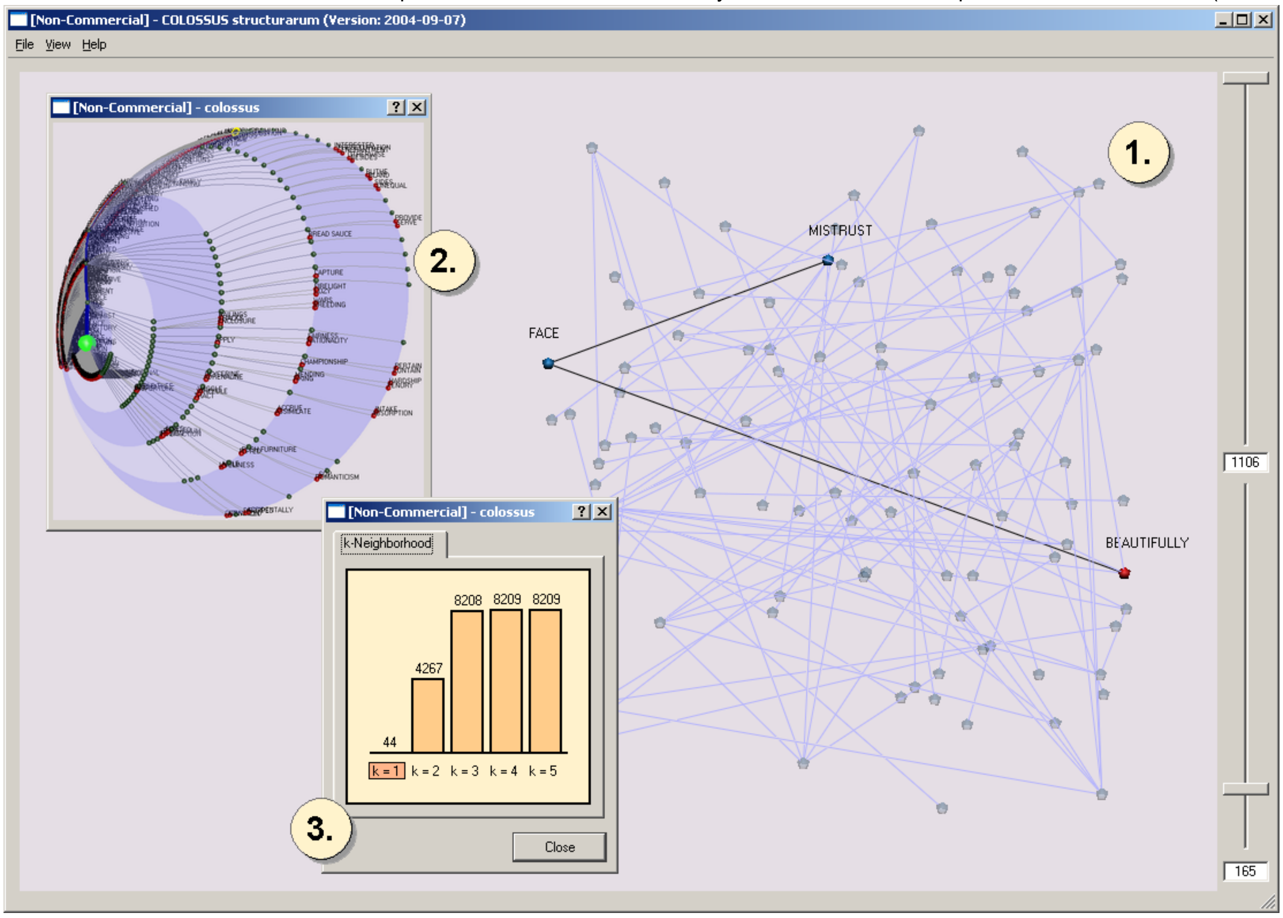

[Non-Commercial] - COLOSSUS structurarum (Version: 2004-09-07)
File View Help
[Non-Commercial] - colossus
2.
[Non-Commercial] - colossus
k-Neighborhood
8208 8209 8209
4267
44
k = 1 k = 2 k = 3 k = 4 k = 5
Close
3.
1.
MISTRUST
FACE
BEAUTIFULLY
1106
165

Figure 5: An overview of the framework's GUI: (1.) the content view showing a small part from a larger data set, including a highlighted shortest path and a red colored, selected node; (2.) the navigational tree-view [MagicEyeView (Kreuseler & Schumann 2002)] containing the browsable result of a hierarchical cluster algorithm; (3.) the detailed view of the k-Neighborhood-distribution of the selected node from (1.) with its 1-Neighborhood being selected.

- CRUSHING, DESTRUCTIVE, DESTROYING
- ANTICIPATE, INSTRUCTIONS, AWAIT

Moreover, to establish interesting connections between two selected words, the user can interactively select the words, and compute the *shortest paths* between them (arrows 9, 10 and 13). Then, this path can be depicted and *focused* in the *content view* (see fig. 5), showing a path between the words MISTRUST and BEAUTIFULLY), keeping the rest of the graph in the *context* applying alpha blending.

Summarizing, the user has a variety of possibilities to interact with the provided modules. Therefore, our framework delivers a variety of exploration paths, to support various exploration contexts and tasks. The user can refine the focus on the data set and explore substructures (fig. 4, arrow 14), refine exploration context (arrow 15), and then restart the whole process. Thus, as an iterative process, the framework supports alternative visual navigation and mining paths to the desired result (arrow 16).

6 Conclusion and future work

In this paper, we investigated the tight integration of methods from graph theory with visualization methods. Therefore, we introduced graph theoretical methods and their applicability for a VDM of structures systematically. In particular, we described how to apply these methods to design good visual representations. Furthermore, we introduced a general, modular and flexible design for a VDM framework for structures, outlined its implementation details and discussed its applicability based on a real-world example.

We tested our framework with different data sets, for instance a WWW-link-structure with 50.000+ web sites and approx. 425.000 links in between them. The gained results where highly satisfying. Nevertheless, there are still challenges for future work. We have to continue testing and evaluating the framework's usability and its scalability to even larger structures. Additionally, more measures, algorithms and visualization techniques need to be integrated.

Acknowledgements

The authors like to express their thanks to Prof. Andreas Brandstädt for helpful discussions as well as Andreas Pohl and Clemens Nafe for testing, evaluating and using the framework for their research on peer-to-peer networks.

References

Abello, J., Finocchi, I. & Korn, J. (2001), Graph Sketches, *in* 'IEEE Symposium on Information Visualization (InfoVis'01), San Diego', pp. 67–72.

Abello, J. & van Ham, F. (2004), Matrix Zoom: A Visual Interface to Semi-external Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 183–190.

Ahlberg, C. (1996), 'Spotfire: an information exploration environment', *SIGMOD Record (ACM Special Interest Group on Management of Data)* **25**(4), 25–29.

Ankerst, M. (2001), Visual Data Mining with Pixel-oriented Visualization Techniques, *in* 'Proceedings of ACM SIGKDD Workshop on Visual Data Mining'01; San Francisco'.

Batagelj, V., Mrvar, A. & Zaveršnik, M. (1999), Partitioning Approach to Visualization of large Graphs, *in* 'Proceedings of the 7th International Graph Drawing Symposium', number LNCS 1731, pp. 90–97.

Bertin, J. (1981), *Graphics and Graphic Information-Processing*, Walter de Gruyter.

Brandes, U. (2001), 'A Faster Algorithm for Betweenness Centrality', *Journal of Mathematical Sociology* pp. 163–177.

Brandes, U. & Corman, S. (2002), Visual Unrolling of Network Evolution and the Analysis of Dynamic Discourse, *in* 'IEEE Symposium on Information Visualization (InfoVis'02), Boston', pp. 145–151.

Brandes, U. & Wagner, D. (2003), visone - Analysis and Visualization of Social Networks, *in* 'Graph Drawing Software', Springer, pp. 321–340.

Bunke, H. (2000), Graph matching: Theoretical foundations, algorithms, and applications, *in* 'Proc. Vision Interface 2000, Montreal', pp. 82–88.

Edachery, J., Sen, A. & Brandenburg, F. (1999), Graph Clustering using Distance-k Cliques, *in* 'Proceedings of the 7th International Graph Drawing Symposium', number LNCS 1731, pp. 98–106.

Fekete, J.-D., Wang, D., Dang, N., Aris, A. & Plaisant, C. (2003), Interactive Poster: Overlaying Graph Links on Treemaps, *in* 'IEEE Symposium on Information Visualization (InfoVis'03), Seattle'.

Fequete, J.-D. (2004), The InfoVis Toolkit, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 167–174.

Frischman, Y. & Tal, A. (2004), Dynamic Drawing of Clustered Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 191–198.

Fruchterman, T. & Reingold, E. (1991), 'Graph drawing by force-directed placement', *Software – Practice and Experience* **21**(11), 1129–1164.

Gansner, E., Koren, Y. & North, S. (2004), Topological Fisheye Views for Visualizing Large Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 175–182.

Girvan, M. & Newman, M. (2002), 'Community structure in social and biological networks', *PNAS* **99**(12), 7821–7826.

Granitzer, M., Kienreich, W., Sabol, V., Andrews, K. & Klieber, W. (2004), Evaluating a System for Interactive Exploration of Large, Hierarchically Structured Document Repositories, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 127–133.

Heer, J., Card, S. K. & Landay, J. A. (2005), Prefuse: a Toolkit for Interactive Information Visualization, *in* 'CHI 2005, Human Factors in Computing Systems'.

Herman, I., Marshall, M. & Melançon, G. (2000), Automatic generation of interactive overview diagrams for the navigation of large graphs, Technical Report INS-0014, Reports of the Centre for Mathematics and Computer Sciences.

Kiss, G., Armstrong, C., Milroy, R. & Piper, J. (1973), An associative thesaurus of English and its computer analysis, *in* 'The Computer and Literary Studies', Edinburgh University Press.

Kreuseler, M., Nocke, T. & Schumann, H. (2004), A History Mechanism for Visual Data Mining, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 49–56.

Kreuseler, M. & Schumann, H. (2002), 'A Flexible Approach for Visual Data Mining', *IEEE Transactions on Visualization and Computer Graphics* **8**(1).

Lamping, J., Rao, R. & Pirolli, P. (1995), A focus+context technique based on hyperbolic geometry for viewing large hierarchies, *in* 'ACM Proceedings of Computer-Human Interaction (CHI95); Denver, Colorado, USA', pp. 401–408.

Nocke, T. & Schumann, H. (2004), Goals of Analysis for Visualization and Visual Data Mining Tasks, *in* 'CODATA Workshop Information, Presentation and Design (March 2004), Prague'.

Robertson, G., Mackinlay, J. & Card, S. (1991), Cone trees: Animated 3d visualization of hierarchical information, *in* 'ACM Proceedings of Computer-Human Interaction (CHI'91)', pp. 189–194.

Roth, S. A., Lucas, P., Senn, J. A., Gomberg, C. C., Burks, M. B., Stroffolino, P. J., Kolojejchick, J. A. & Dunmire, C. (1996), Visage: A User Interface Environment for Exploring Information, *in* 'IEEE Symposium on Information Visualization (InfoVis'96), San Francisco', pp. 3–12.

Scharl, A. (2002), Adaptive Web Representation, *in* 'Human Computer Interaction Development & Management', pp. 255–260.

Shi, J. & Malik, J. (1997), Normalized Cuts and Image Segmentation, *in* 'Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'97)', pp. 731–737.

Shneiderman, B. (1992), 'Tree Visualization with Treemaps: A 2D Space Filling Approach', *ACM Transactions on Graphics* **11**(1), 92–99.

Stolte, C., Tang, D. & Hanrahan, P. (2002), 'Polaris: A system for query, analysis, and visualization of multidimensional relational databases.', *IEEE Trans. Vis. Comput. Graph.* **8**(1), 52–65.

Tollis, I., Eades, P. & di Battista, G. (1999), *Graph Drawing - Algorithms for the Visualization of Graphs*, Prentice Hall.

Valiente, G. (2002), *Algorithms on Trees and Graphs*, Springer.

van Ham, F. & van Wijk, J. (2004), Interactive Visualization of Small World Graphs, *in* 'IEEE Symposium on Information Visualization (InfoVis'04), Austin', pp. 199–206.

Voigt, D. (2001), WWW-based Representation of complex Information Structures (in German: WWW-basierte Darstellung komplexer Informationsstrukturen), Master's thesis, University of Rostock, Department of Computer Science.

Zhang, P. (1994), *Method of Mapping DNA Fragments*, United States Patent No. 5667970.

Shallow NLP techniques for Internet Search

Alex Penev **Raymond Wong**

National ICT Australia and School of Computer Science and Engineering,
University of New South Wales, Sydney, NSW 2052, Australia
{alexpenev,wong}@cse.unsw.edu.au

Abstract

Information Retrieval (IR) is a major component in many of our daily activities, with perhaps its most prominent role manifested in search engines. Today's most advanced engines use the keyword-based ("bag of words") paradigm, which concedes some inherent disadvantages. We believe that natural language (NL) is a more user-oriented, context-preservative and intuitive mechanism for web search.

In this paper, we explore shallow NLP techniques to support a range of NL queries over an existing keyword-based engine. We present JASE, a web application enveloping the Google search engine, which performs web searches by decomposing input NL queries and generating new queries that are more suitable for the search engine. By using some of Google's syntactic operators and filters, it creates "clever" queries to improve precision.

A preliminary evaluation was conducted to test JASE's accuracy, and results have been encouraging. We conclude that the NL model has potential to not only rival the keyword-based paradigm, but substantially surpass it.

Keywords: Information Retrieval, Natural Language Processing, Google.

1 Introduction

At present, the holy grail of IR is embodied in the World Wide Web—an ever-growing source of self-updating information that is easy to access yet difficult to discover.

Today's engines use the keyword-based paradigm, by implicitly connecting given keywords with boolean operators (and, or, not). This model concedes certain inherent disadvantages that are becoming increasingly evident as the web continues to expand—context is lost once keywords are isolated and treated on an individual basis, and many words carry double-meanings. Together, these deficiencies result in larger recall which is filled with noise, frustrating the user.

We believe that natural (or *everyday*) language is the ideal mechanism for information discovery—it is user-oriented, because it is intuitive and requires no training. It allows users to express a query in the way that it is rationalized and constructed in their mind, while both providing a context and helping to disambiguate word senses. But NL queries such as "*german or austrian composers born in the 1600s*" and "*native animals in australia, but not marsupials*" show that there is room for improvement in today's engines. More elaborate queries such as "*700ml Johnnie Walker Red Label, in Sydney for under $30*" cannot be answered at all, even when appropriate documents are indexed by the engine.

Through practice and tribulation, users learn to mentally cull their queries into a set of only the few "most important" components, before sending a request to an engine. This structural rearrangement, coupled with search engines' treatment of keywords as individuals, could be impeding their accuracy.

Furthermore, to become proficient at using a certain engine, users must learn its special operators. These differ between each engine and render the search mechanism to be unnatural, due to the introduction of foreign modifiers into the query. Our evaluation survey indicates that average Google users are largely unaware of its operators and filters, and rarely use them in practice.

There has been few significant advances in Internet search for half a decade, and the shortcomings of the keyword-based paradigm are likely to be globally costing millions of hours each year in labor for wading through voluminous results. Meanwhile, the Internet continues to grow and permeate our way of life, and search results become larger and potentially noisier. Therefore, we believe that NL will play a principal role in web and media search in the near future, because it is more intuitive and provides more information than the current model.

In this paper, we explore shallow NLP techniques to support NL queries over Google. We evaluate the performance and accuracy of JASE, an application enveloping Google, which decomposes NL queries to form Google-friendly queries and reranks the retrieved results. We define a categorical classification of searchable entities and highlight how they can be used in conjunction with Google's advanced operators and filters. We propose heuristics that can be used for the reranking step. Finally, we assess our system's performance for a set of keyword and NL queries.

The remainder of this paper is organized as follows; §2 provides an overview of search engines, and defines our problem domain. §3 outlines the algorithmically-disparate phases and data structures of our system (explored in greater detail in §4 and §5). We conduct a comparison of the accuracy of our system against Google and "average Google users" for a mix of keyword and NL queries in §6. §7 covers related work, and §8 concludes this paper.

2 Background

As JASE is a wrapper for a search engine, it is vital to understand how search engines work. This allows us to determine which subprocesses are to be implemented by JASE and which are delegated to Google.

2.1 Search Engines

A search engine is an online program which, for a given query, retrieves references to web documents that match it. In theory, a search engine has four components:

document processor indexes new documents. Indices are a mapping between words and what documents they appear in. Most engines are spider-based, so a crawl of the web for new documents and the updating of the index is automated.

query processor inspects a user's query and translates it into something internally meaningful.

matching function uses the above internally meaningful representation to extract documents from the index.

ranking scheme positions the more-relevant documents on top, using some relevance measure.

Users communicate with the query processor, which is the only visible component. It carries out several tasks, usually (but not limited to):

- tokenizing of the query to remove invalid characters, and to recognize meta-keywords or special syntactic operators.
- removal of stopwords; words which are too common and rarely help in the search (e.g. the, a, of, to, which).
- stemming; a process designed to improve the performance of IR systems, involving normalizing semantically similar words to their root forms (e.g. produce, produced, producer, producers, produces and producing map to *produc-*).
- assigning a weight to each keyword/keyphrase, to aid with ranking(Salton & Buckley 1988).

After results are retrieved by the matching function, they are ranked by relevance based on some ranking measure and set of heuristics (called the ranking scheme). Often taken into account are:

term frequency how many times keywords appear in the document(Luhn 1957).

inverted document frequency a value which aims to determine how important a term is in discriminating a document from others(Salton 1989, Jones 1972).

semantic proximity words synonymous to a given keyword may be matched, boosting the score of the document.

term position keywords appearing in the title or heading (rather than the body) should contribute more to a document's weight.

term proximity a document in which the query terms are close together is considered more relevant than one in which they are far apart.

cluster distance how far apart groupings of matched terms are.

percentage of query terms matched.

In our case, JASE implements the query processor and the ranking scheme, while Google provides the document processor and matching function. Of course, Google utilizes its own query processor and ranking scheme for any query that it answers, and therefore JASE's results will be heavily influenced by Google's own relevance measure. We exercise some indirect control over these components, since JASE's query processor is invoked before and JASE's ranking scheme is invoked after Google's.

2.2 JASE

The initial design decisions for our system were that JASE will be a web application into which a user enters free text (preferably NL) in a query box, presses a button, and views the corresponding search results. It should implement the query processor and ranking scheme components of a search engine, so that it can influence what was being sent to Google, and influence what was being relayed back to the user.

Behind the scenes, JASE would invoke Google via an API[1]. Google is not designed to handle NL queries, so JASE would have to manipulate the input to make it Google-friendly. It would also take advantage of the syntactic operators and filters to try and improve precision. JASE would display the same information that Google displays—a title, URL and snippet for each matched document. We had decided not to collect any further information regarding documents, such as a downloading of the actual source.

2.3 Google's API

Google (the company) provides many online services, most important of which is Google (the search engine). This engine is useful for JASE because it:

- advertises a free API (over WSDL/SOAP), allowing it to be remotely queried from many programming languages and environments.
- provides the title, URL and snippet of matched documents, which serve as a *synopsis.*
- has powerful operators and filters: +, -, ~, OR, intitle:, inurl:, site:, filetype:, numerical ranges, timestamps, related:, link:, and some limited wildcard matching.
- performs stemming.
- is case-insensitive.
- ignores most punctuation.
- uses sophisticated link- and structure-based analysis to determine the importance of documents on both global (e.g. PageRank(Page, Brin, Motwani & Winograd 1998)) and per-query scales (e.g. anchor text, keyword proximity, and whether or not keywords appear emphasized with markup in the document).

Some limitations of the API are that a maximum of 10 terms can be sent per query, that languages other than English are poorly supported, and that only 10 results can be retrieved per query. Case-insensitivity is a pro because it allows users to be sloppy, but also a con because acronyms and proper nouns will sometimes match unrelated documents. The API is still in the beta stage, and its functionality may be altered at any time.

These points affect the methods and messages that JASE can use to communicate with Google. Obviously, we cannot, for example, perform web search

[1] http://www.google.com/apis

using regular expressions, because the engine does not offer such functionality.

2.4 Problem Domain

One of the challenges of applying NLP over English is that the language allows many syntactic variations of sentences. Semantically identical questions can usually be worded in more ways than one: *how old are you?*, *what is your age?*, *which birthday did you celebrate this year?*. Questions with logical constraints can be permuted even further; for example, with composer as subject, and logical constraints on a date of birth and nationality, we can ask for "Which composers were born in Germany or Austria between 1600 and 1700?", or "What are the names of some German or Austrian composers born in the 1600s?", or "In the 17th century, what famous composers were born in either Germany or Austria?", and many others. The wording of the query will differ between people, but all of these will have somethings in common—a composer, a nationality German/Austrian, and a year of birth. These are the entities that we wish to extract, whereby two variations on the same query will yield the same decomposition.

It is important that we restrict the types of queries we want to handle, as the general query domain is too overwhelming. We focus on a subset of all possible query structures, to roughly satisfy the grammar:

- `P = ( K | Y | M )`
  where `K` = some keywords that form a phrase
  `Y` = year phrase
  `M` = money phrase
- `PS = P ( connective?  P )+`
  Two or more disjoint runs of phrases. The connectives will mostly be conjunctions, prepositions or adverbs.
- `( P | PS ) negation+ P`
  Things that the user does not want, e.g. "*[American presidents] [except] [Bush]*". Negations are usually negative coordinating/correlative conjunctions or adverbs.
- `( P | PS ) ''in C'' (P | PS )?`
  where C is some location, e.g. "*[composers born] [in Germany]*". In general, words that follow *in* will rarely be locations, so we must explicitly provide a list of permissible matches. We can use such matches to focus queries towards certain domains, such as *.de*, for the given example.

Some typical examples of the atoms include:
`K` = *jujitsu*
`Y` = *1920*, *500 BC*, "*during the 90s*"
`M` = *$60*, "*cheaper than $10*", "*between $5 and $10*"
`PS` = "*[endangered animals] in [australia]*", "*[composers born] [during the 1600s]*".

The above is not intended as a formal grammar, but as a guide to visualize possible queries. JASE still perform a best-effort search, irregardless of whether a query satisfies the above grammar.

3 System Overview

Our system is active at the beginning and end of the search session and we follow a typical Service Oriented Architecture, with a consumer (JASE) and provider (Google). Consequently, we are able to partition the functionality in two disparate phases.

Phase One JASE acts as the query processor. The search query is parsed and decomposed. Decomposition involves tokenizing the text to discover structural objects, such as words, numbers and punctuation. These must be stored as some internally-meaningful representation, which we call the **SearchTerms**. The contents of this container are the seed for generating new queries.

Phase Two JASE acts as the ranking scheme. In Phase One, variants of the input query are submitted using the API calls, and each retrieves up to 10 results. This leaves us with many "best" ranked documents, as each result set has its own top match (note that some result sets are likely to overlap). We must perform a reranking on the results as a whole, based on some relevance measure. We rerank by assigning a score to each document and sorting. Our heuristics for calculating a document's score are described in §3.2.

3.1 SearchTerms, a link between Phases

At the beginning of Phase One, useful information is extracted from the query, grouped and then classified in the SearchTerms container. In Phase Two, retrieved documents are compared against these SearchTerms to receive a score. This container is the connection between the phases, which are otherwise independent. The container is composed of several sets of "searchable entities", which we define as:

input query as a unit phrase.

primary keyphrases extracted keywords and keyphrases. Phrases are the most specific and discriminatory part of queries, thus adjacent keywords should be grouped as a phrase wherever possible. A phrase may contain a singleton word, if it happens to be bounded at both ends by non-keywords. This set is never empty.

secondary keywords primary keyphrases are broken down into their individual atomic keywords, each becoming a secondary keyword. This set may be empty, since it will not contain keywords that are already primary.

tertiary words any remainder terms from the original query, which have not been categorized as primary or secondary are inserted here. This set will largely consist of stopwords.

synonyms/hyponyms/meronyms for singleton primary/secondary words come from an external source, such as WordNet[2].

- synonyms are words with corresponding meaning, e.g. alcohol/liquor.
- hyponyms are more-specific words, e.g. dog/poodle.
- meronyms are parts of a larger whole, e.g. dashboard/car.

exceptions are undesirable matches; things the user does not want.

Two additional pieces of information are also recorded. The first is a list of numerical upper and lower bound tuples, used to apply numerical range matching. As a proof-of-concept, JASE supports dates and prices, but other ranges are possible to detect and match. The second datum that we record is a domain restriction, in order to restrict some queries to a specific domain. At present, we handle mappings

[2]http://www.cogsci.princeton.edu/%7ewn/

from the ISO 3166[3] list, with a few obvious adjustments (e.g. removal of *.us*).

3.2 Weighting

The SearchTerms sets are assigned base weights, representative of the desirability of their inclusion. Each document in a result set is assigned a score, which is derived by comparisons against the SearchTerms. When grading a document, we consider only the title, URL and snippet—no extra information about documents is obtained. Using only the title/URL/snippet allows us to evaluate JASE's performance and accuracy using information which is made visible by the search engine. Digging deeper and downloading the source of documents may lead to more accurate scoring, but is outside our scope.

As matching a phrase should be more desirable than matching only one word of that phrase, one would expect phrases to contribute more than words. The location of the match (title/URL/snippet) also affects its contribution. Table 1 defines the base weights for the SearchTerms. These values are subject to tweaking as there is no correct answer, but they seem to work well in practice. Certain matches may appear in more than one set, in which case the higher weight is used. We also use stemming, case-mismatch and term frequency (see §5.1).

Entity	**title**	**URL**	**snippet**
Input query	3.0	3.0	1.2
Primary	1.0	1.0	0.4
Secondary	0.5	0.5	0.2
Synonym	0.3	0.3	0.12
Hyponym	0.3	0.3	0.12
Meronym	0.3	0.3	0.12
Tertiary	0.2	0.2	0.08
Exception	-10	-10	-4

Table 1: Base weights of SearchTerms entities

4 Phase One

This phase involves the parsing and tokenizing of the input query, to build the SearchTerms container. Extracting the most sensible keyphrases and keywords from the user's NL query is critical, as Google is a keyword-based engine and its results will heavily fluctuate depending on what terms are chosen. Our strategy is to create "clever" Google-specific queries, which contain several of:

keywords are important and discriminatory words. Keywords can be directly sent to Google.

keyphrases are sequences of keywords, where order is important. JASE detects them by looking for phrasal boundaries. Keyphrases must be quoted to be recognized by Google, e.g. "*vampire bats*".

exceptions are terms the user does not want, often explicitly stated. Both word and phrase exceptions must be preceded with a minus, e.g. "*pets -dogs*".

domain restriction concentrates a search on a particular domain, e.g. "*national park site:nz*".

synonymy refers to terms which are related to a keyword, but they need not be specified by the user. We are only interested in three categories: synonyms, hyponyms and meronyms. Others categories exist (such as hyper/anto/pertai/holonyms), but are not useful for this task. Google will look for some synonyms if a word is preceded by a tilde; for example, the search "*~movie -movie*" matches *video*, *film*, *dvd*, *mpeg*, *cinema*, *soundtrack* and *trailer*. JASE does not perform query expansion using synonymy, but it does look for them when scoring a document.

numerical ranges place a constraint for lower and upper bound matches. Ranges of the form *lo*..[*hi*] are supported by the engine, and some units are also recognized (e.g. "*beethoven symphony 8..*", "$50..60" for price and "100..200 kg" for weight).

One of the advantages of automating web search is the ability to fire off many different queries and selecting only the best results. JASE emits between two and twenty new queries for a given search, depending on the complexity of the query, and how many of the SearchTerms sets are utilized. Some previous work (e.g. (Kwok, Etzioni & Weld 2001, Agichtein, Lawrence & Gravano 2001)) has empirically shown that such an immediate increase in recall, despite its overhead, is a very effective. This strategy, however, raises a few issues. Different queries must be sent each time, prompting the need for a mechanism to formulate slightly variant queries, using the SearchTerms as a seed. Each query will also have its own top result, so an equitable reranking mechanism is needed. Finally, some documents are likely to be returned by several queries, therefore must be clustered as one. JASE addresses each of these points.

4.1 Detecting Keywords and Keyphrases

Known techniques for locating phrases in written text tend to use vector-space weighting algorithms, naïve Bayesian classifiers, inverse document frequency (IDF) tables, lexical chains, or other statistical means. However, these are all intended to be applied to whole documents, and are trained on a specific corpus. In contrast, we are dealing with a single line of input, ranging from one to maybe fifteen words. Such confined input makes it difficult to use statistical models, especially since many phrases will contain proper nouns and not be found in any corpora. From these, we feel that an IDF table is the only suitable approach.

Another NLP technique is to deduce the parts of speech using a Part Of Speech tagger. These can be rule-based(Brill 1992, Brill 1995) and follow patterns, unigram or n-gram based, or Hidden Markov Model based(Charniak 1994, Charniak 1997, Collins 1996) and follow probability. The tagger can be used to detect phrases by collecting disjoint runs of nouns and adjectives. Many taggers exist, but JASE does not use one. Instead, JASE guesses the location of phrasal boundaries by splitting on stopwords, using our own custom 99-word list (a hybrid of Google's and Snowball's[4], with additions). Since stopwords are those with a poor IDF, this strategy emulates the use of an IDF table to some degree.

This means that the "naturality" of the language used is important for adequately deducing phrases. On par with previous NL systems that we have played with, it is not difficult to construct complicated but unnatural-sounding queries to confuse JASE. For our evaluation (§6), we used sensibly-worded queries. Our shallow NLP approach works well for many of the TREC queries, and using deeper NLP will only serve to improve accuracy.

[3]http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2

[4]http://www.snowball.tartarus.org

4.2 Detecting a domain restriction and numerical ranges

JASE pattern matches the user's query for countries. If preceded by *in*, *in the*, *from* or *from the*, it will focus approximately half of the generated queries towards that country's domain. Most country domains have some web pages written in English, so it not unreasonable to carry out such searches.

JASE also pattern matches year and monetary phrases, to demonstrate how numerical ranges can be extracted. It handles many cases, best illustrated by some examples (Table 2). Date matching is capped at year 9999. If a lower or upper bound of a monetary range is not specified, JASE assumes 10% for the lower bound and a string of nines to one more significant figure for the upper bound.

Phrase	Range
Rock stars in the 60s	1960..1969
Rock stars in their 60s	60..69
Rock stars before the 60s	0..1959
Wars before 1066	0..1065
Wars during 1066	1066..1066
Roman emperors before 20 BC	21..99
A Kodak camera cheaper than $200	$20..200
A Kodak camera, over $60000	$60000..999999
$49.99 Playstation controller	N/A
$49 Playstation controller	$49..50
Playstation controller between $49 and 60	$49..60

Table 2: Some examples of handled ranges

5 Phase Two

This phase deals with the reranking of all query results, using relevance heuristics against the SearchTerms constructed in Phase One.

All documents are assigned a numerical score (or weight) based on the relevance measure. The documents are sorted, and only the top 20 are displayed to the user. As mentioned previously, JASE only uses the synopsis of a document (title/URL/snippet) to weigh a document.

Two morphological processes are carried out before documents are inspected—folding case and stemming. The original synopsis is used for case comparison; case-insensitive match is performed on the lower-case version, and stem matching is performed on the stemmed version. As Google uses stemming, it is possible to encounter a partial match in the document synopsis, which needs to be rewarded. We use a Porter stemmer(Porter 1980).

5.1 Relevance Measure and Heuristics

Upon meeting a document in the result set, its synopsis scanned for entities within the SearchTerms sets. As described in §3.2, certain sets are more important than others and contribute different base weights, and the location of a match also affects the contribution. It is furthermore influenced by the nature of the match:

- if a term is indirectly matched via stemming, its contribution is penalized by 25%.
- if case agrees, its contribution is boosted by either 25% if it appears in the title/URL, or 10% otherwise.
- if a term is matched via some numerical range, its contribution is penalized by 25%.
- if every primary and secondary term is matched at least once, the overall document score is boosted by 50%.

These figures are subjective and there is no correct answer, but they work well in practice.

5.2 Overall Score of a Document

A naïve algorithm to determine the total score for a document would be to sum all individual contributions for all SearchTerms entities. If an entity is matched, it contributes its base weight less penalties plus bonuses, and if it is not matched, it contributes nil. This approach suffers the problem that repetitive matching of a single entity, while matching few or even none of the others, results in a score that unfairly represents the document. This weakness becomes more evident with small result sets like JASE's, as they are more volatile to fluctuations in rating. JASE cannot get an "averaging out" effect without retrieving far larger results sets, and the snippet only partially communicates the content of a document.

Clearly, such a biased boost due to multiple matching of a single entity is inappropriate. But at the same time, we do not want to ignore recurring matches entirely, because it is desirable for every match to contribute "something". To deflate the volatility of recurrent matches, we use a recessive geometric sum to calculate the score of a document:

$$score(doc) \quad = \quad \frac{1}{G} \quad + \quad \sum_{e=e_1}^{e_n} \sum_{i=1}^{j} \frac{\omega_i}{2^{i-1}}$$

where $\{e_1..e_n\}$ are the n searchable entities in the SearchTerms. The set of weights $\{w_1..w_j\}$ represent the contributions by the j occurrences of entity e matched in *doc*'s synopsis. This set of weights is sorted in descending order to maximize the overall result.

Such a summation guarantees that every occurrence of a matched entity e contributes to the document, but no entity can contribute more than twice its highest individual match to the overall document score (recall $\sum_{i=1}^{\infty} \frac{1}{2^{i-1}} = 2$). This satisfies our above desiderata that superfluous entity frequency should not bias a document "too much", but that every occurrence of any entity should contribute.

Furthermore, the reciprocal of Google's suggested rank, G, is augmented to a document score. This bonus contributes as much as 1.0 point for Google's topmost result, and is worth the same as a matching of a primary keyphrase in the title. Because this bonus diminishes for lower ranks in Google's list, it is a means of discriminating between documents that receive similar scores against the SearchTerms, but appear in different ranks in the original results.

5.3 Reranking Documents

The previous section describes how a score is assigned to a document. However, some queries may overlap and documents may be met more than once. We do not wish to display the same document several times over, so the scores for a certain document must be accumulated. Possibilities include a running total, an average, or a recessive geometric sum like the previous section. We use the geometric sum because it is a middle ground between the running total and the average—two methods which work well in most cases, but spectacularly fail for some scenarios.

Once documents are reordered, the top 20 are displayed to the user.

6 Evaluation

This section serves as a summary of our system's accuracy and performance. All results in this section were carried out during Oct 2004.

6.1 Accuracy

An evaluation survey was prepared, and answered by 8 volunteers. All were fluent in English, and all confirmed that they use Google at least once per week, with more than half using it daily. Participants were of mixed age (18–49), mixed gender and mixed nationalities (USA, Australia, UK, France and Switzerland). One participant had a computer science background. We feel that this sample represents "average" Google users to a reasonable degree.

Participants were first asked about their searching prowess, shown in Table 3.

Question	Yes	No
Do you know of operators: +, -, ""?	75%	25%
Do you use them?	38%	73%
Do you know of filters: site, inurl, intitle?	-	100%
Do you know of indirect matchers: ~, x..y?	-	100%

Table 3: Supplementary questions answered by participants

While most knew of the plus, minus and quotes operators—used for inclusion, exclusion and phrases—only half of those admitted to actually using them in practice. A small proportion were familiar with the *site:* filter only, but none of the other operators.

Our survey contained 14 query topics in roughly increasing complexity, which are listed in Table 4. Some queries consisted of only keywords, while others were written in NL. We opted to avoid relying on the TREC test set, because many web documents explicitly quote the TREC queries in the context of TREC, yet are unrelated to the topic at hand. Only q6–q9 are TREC queries.

Id	Query
q1	microsoft
q2	belgian comic strip characters
q3	endangered animals in australia
q4	what happened at the final in the 2002 world cup?
q5	German or Austrian composers born in the 1600s
q6	what is the treatment for alzheimer's?
q7	how much sugar does Cuba export and which countries import it?
q8	the consequences of implantation of silicone gel breast devices
q9	what diseases have hair loss as a symptom?
q10	important discoveries in medicine during the 1600s
q11	A used Toyota Camry 1998 model, in Sydney between \$5000 and \$10000
q12	700ml Johnnie Walker Red Label in Sydney for under \$30
q13	Famous people born on May 1 between 1900 and 1950
q14	I want to do Artificial Intelligence in the best university in Australia

Table 4: Base weights of SearchTerms sets

A scale from 1 to 5 was defined (1 being poor, 5 being excellent). For each query, the top ten Google[5] results were provided. Participants were asked to rate the result set based on their impressions and opinions in regard to accuracy. Participants were then asked to perform their own search using the search engine, and could rewrite the query in any way using any tactics they wished. Their goal was to find relevant documents that satisfied the query. They then proceeded to rate their own results. Finally, participants rated JASE's results for the original query. In all cases, participants were encouraged to view the actual documents retrieved by visiting the hyperlinks.

JASE received favorable ratings, outperforming Google for some queries, while being approximately equal for the remainder. This result was not unexpected—for simple keyword queries, JASE displays almost exactly what Google does, but for more complex queries, its Google-optimized queries and reranking appeared to improve accuracy.

But as one of the aims of this work was to show that NL queries can be used to improve the precision of web searches, we were more interested in how JASE would fare against the participants themselves. Figure 1 shows how each of Google, the participants and JASE performed for each query. A 95% confidence interval is provided, using the *t*-distribution.

Google's Accuracy
This form of web search represents "unskilled" searching. Participants rated the Google's raw search results for each verbatim query. Simple keyword-only queries received high scores, but ratings gradually fell as queries became more complex and involved NL. This behavior was expected.

Participants' Accuracy
This form of web search represents "semi-skilled" searching. Participants used the given query to create their own new query based on their experience and knowledge of the search engine, and evaluated the results. Observed tactics included phrasal search, domain restrictions, and query expansion. As expected, participants were able to slightly outperform the "unskilled" search.

JASE's Accuracy
This form of web search represents "skilled" searching. Participants rated JASE's search results for each verbatim query. JASE kept up with both unskilled and semi-skilled searches for the simpler queries—which were mostly keyword-based to test the accuracy of JASE's reranking—but maintained a lead for the second half of queries, which were written in NL and highlighted JASE's advantage of dispatching multiple queries and usage of an engine's filters and operators.

One observation is that JASE's confidence interval does not overlap the others for some of the NL queries. According to the *t*-test, this is a statistically significant result. The queries that caused this (q5, q10, q12 and q13) highlight JASE's advantage of using Google's filters and operators (the ones used here were "", +, ~ and numerical range). These queries shared in common a need to match text which was implied, but not explicitly stated. Most other queries did not have such implications, and relied on direct keyword-matching. JASE received high ratings there too, most likely due to its increase in recall by retrieving results from multiple queries, and subsequent reranking of results. This conclusion is consistent with previous works (see §7).

The results indicate that JASE outperformed the users themselves for this query set, which involved queries of various difficulty. This suggests that NL queries may be used to improve the accuracy of web search, through shallow NLP systems such as JASE.

[5]http://www.google.com

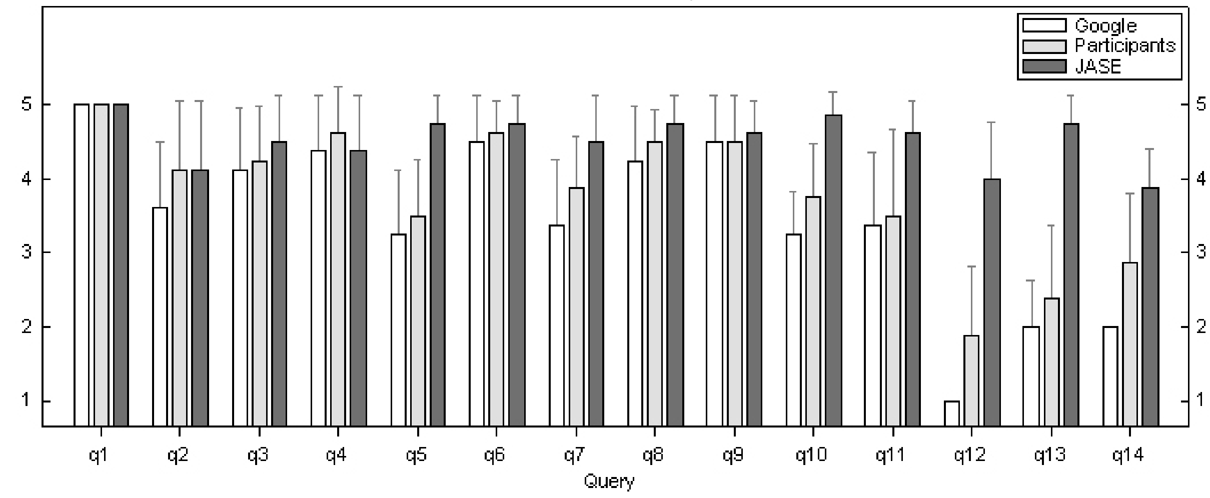


Figure 1: Mean rating per query, with 95% CIs

While expert users may be able to match or surpass the precision of such a system, the system should be beneficial to average users, who are unfamiliar with the esoteric art of accurate web search.

6.2 Performance

As a synchronous wrapper around Google, JASE is inherently slower than Google. There are three separate time periods involved from the moment a user inputs their query for processing until the moment the results are displayed:

GT Google time, consumed by the search engine in answering our queries. It is conveniently reported by the API methods.

CT Communication time (`round trip time - GT`). Timing begins with the invocation and return from the SOAP library routines.

JT JASE time, time during which JASE's code is active, including bootstrapping of any libraries and classes.

The time periods are mutually exclusive, and the total time taken is `TT = JT + CT + GT`. Depending on the complexity of the input, JASE emits up to about 20 queries in total.

Our empirical tests indicated that `JT` $\approx$ `0.11 TT`, `GT` $\approx$ `0.21 TT` and `CT` $\approx$ `0.68 TT`. The biggest performance cost, `CT`, represents networking and communication. The tests suggested that our prototype was not inefficient, as the bulk (89%) of loading time was consumed externally. Our prototype took just under 20 seconds to load for the more complicated queries, because we submitted queries serially. This value may be greatly decreased by issuing queries in parallel, but speed was not our goal.

Participants in our evaluation were asked their opinions on search response times. All agreed that an extra 5 seconds wait on top of Google's average response time (which is between 0–0.5s) in order to produce more accurate results is admissible. In fact, 75% agreed that even 20s was admissible. Such an answer hints at web users' desideration for a system such as JASE.

7 Related Work

Many NLP search systems have been made to date. In particular, a form of NL search called Question-Answering Systems have been well-explored (e.g. (Katz 1991, Kwok et al. 2001, Prager, Brown, Coden & Radev 2000, Ravichandran & Hovy 2002)).

QAS accept wh-questions (who, when, where, what, why) and return a definitive answer. QAS are related to search engines because they retrieve information from a source based on a query. Unlike search engines, QAS provide an answer, rather than a list of top "hits". To do this, QAS need to have at least some idea of what the user is searching for. As such, QAS usually extract knowledge from the query itself—is the user asking for a person, a date, a location, an object, or what? On the other hand, search engines use whatever input they have been given with minimal restrictions on format and structure. QAS have much stronger restrictions on the structure of the input, in order to make it possible to determine what the user is looking for.

START(Katz 1991) was the first online QAS, and focused entirely on geography and MIT-specific knowledge. It used subject-*relation*-object tuples to extract the subject, relation and object from a given query, and then performed a pattern match for the tuple in its knowledge base (KB). The knowledge base was built from a similar process of detecting tuples from scanned documents. START's KB was highly-edited, and non-scalable, and the system could not provide an answer to a query if it failed to match the tuple. Future work(Katz & Lin 2000) found ways to automate the expansion of the KB, but the process was impractically slow.

Search engines, on the other hand, provide references to documents, irregardless if they answer the user's questions or not, and rarely try to "understand" the query.

Ionaut(Abney, Collins & Singhal 2000) was an interactive NL search engine, and used a local cache of documents for its KB with a small coverage. Its most interesting feature was to list related hyperlinks to a given query to branch into different queries, as a means of an iterative search. From experience with it, we feel that its accuracy was lacking, but the interactivity feature was its best asset.

Limited KB coverage is a hurdle that can be overcome. In 1993, MURAX(Kupiec 1993) used boolean searching over an online encyclopaedia, by formulating queries based on the phrasal content of the input wh-question. Noun-phrase hypotheses were extracted from the retrieved results, and new queries were independently made to confirm the hypotheses. Its accuracy was poor, but the concepts of multiple queries and formulating different queries inspired some future works.

Tritus[6] was an NL search engine(Agichtein et al. 2001) that could use either Google or Altavista as its KB. It handled simple wh-questions that matched specific templates, but used the engines' syntactic operators to improve precision. The authors performed comparisons between different engines, and argued that Google was superior to both Jeeves and Altavista, and that Tritus' Google-optimized queries outperformed raw Google. We consider their testing to be inconclusive, as it is unfair to pipe direct NL queries to a keyword-based engine. An evaluation of this type must involve the users of the engine themselves, who know better than to submit NL queries; the users' employment of the engine should be the real competitor. One further difference between Tritus and JASE is that our work attempts to handle a broader range of input (by not using templates), and that JASE received no training.

Also in 2001, MULDER(Kwok et al. 2001) tried to scale QA to the Web, using Google as back-end. Like MURAX and Tritus, it generated multiple new queries from the input, to increase recall. Query generation was achieved by rearranging the input wh-question to match the potential phrasing of its answer—when asked "what is the capital of Sudan", it would look for documents containing "the capital of Sudan is". MULDER worked on the assumption that the Web is host to more truths than falsities, therefore wh-questions could be answered by collating the results and clustering them. The largest cluster was considered as the correct answer, due to the original assumption. MULDER used deep NLP, but imposed structural limitations on its input queries. We have not had the opportunity to test it, as it has not been available for several years. We believe that MULDER would have performed well for trivia questions because they are frequently cited online, but that it would have had difficulty in answering more elaborate queries such as those in the introduction, for which a rearrangement of the query is unlikely to appear in any online documents.

AskJeeves[7] is advertised as a QAS, but its menu-driven dialogue is more inherent to search engines. It allows searching for both keywords and wh-questions. To answer a NL question, the text must match one its question templates; otherwise, web results are retrieved from Teoma[8]. If a template is matched, AskJeeves provides links to authoritative sites which are known to answer that question. This strategy requires human editors to map the templates to the authoritative sites, and does not scale well. It takes little effort to formulate a NL wh-query which fails to match a template, yet is competently answered by a raw Google search. (Kwok et al. 2001) empirically argues that AskJeeves is limited and awkward to use, and performs poorer than Google.

Intermezzo(Flank 1998) used NLP techniques to retrieve images from an image database based on NL queries, achieving a precision of almost 90%. The content of each image was identified via captions, which were manually written. One of Intermezzo's interesting features was using WordNet to match related terms in the caption to increase the score of an image. This strategy proved effective as the images show physical objects, which have large collections of applicable hypo/hypernyms and mero/holonyms. Such a strategy of boosting term weights using WordNet's synsets is less effective for web queries over large document collections, since matching hyper- and holonyms is less appropriate.

[6]http://tritus.cs.columbia.edu
[7]http://www.ask.com
[8]http://www.teoma.com

Keyword-extraction has a long history and is a component in most IR fields. Several recent approaches to deducing the keyphrases in a piece of text exist (Turney 1999, Turney 2000, Munoz 1996, Frank, Paynter, Witten, Gutwin & Nevill-Manning 1999). However, these methods are intended to extract phrases from entire documents by employing holistic statistical models, while we are interested with extracting useful words and phrases from a single-sentence query. Hence, the algorithms from such works do not apply.

JASE shares the ideas of many of these previous works, such as using an NL wrapper around a boolean data source, and the submission of multiple queries. Because it is not a QAS, it is fundamentally different to MURAX and MULDER in that it imposes less restrictions on input, but therefore cannot use the query structure to its advantage. Tritus was a hybrid, retrieving hyperlinks like a search engine, but handling wh-questions and using the structure of the input to its advantage like a QAS.

Our work aims to be a search engine, but without being able to significantly rely on the structure of the query. The work presented in this paper is most closely related to MULDER and Tritus.

8 Conclusions

This paper has outlined some simple strategies to support NL queries over a keyword-based engine (Google). We have presented an evaluation of a search engine wrapper, JASE, that handles both keyword and NL queries. We parse, tokenize and extract searchable entities from the query, and categorize them into weighted sets. We dispatch multiple queries, and then use the sets against our ranking heuristics to weigh and rerank the retrieved results. Although only shallow NLP techniques were used, they seem work well for many cases, as indicated by our evaluation. Our evaluation survey furthermore revealed that our participants were all willing to sacrifice a large amount of performance in lieu of accuracy, therefore the submission of multiple queries is a justifiable strategy, and may be incorporated in current engines.

Future expansions that we are exploring include deep NLP—a tagger(Brill 1992, Brill 1995) coupled with an IDF table will greatly improve phrase boundary detection. Geographical locations can be detected using a gazetteer, allowing domain restrictions to be used more liberally. Finally, collecting multiple snippets for the top few documents should help improve reranking. This is our preferred way of expanding a document's summary (as opposed to downloading the entire document from the Google Cache), and is achieved by using a combination of *site:*, *allinurl:* and *allintitle:*, coupled with an extra keyword to variate the snippet.

The process of parsing and tokenizing the input to detect important searchable entities is obvious, as these are tasks that the human mind perform when presented with a query topic. But strategies such as submitting multiple queries and reranking of large result sets are only fit for computers. Our evaluation survey revealed that average Google users seldom use its operators and filters, which could be adding noise to their searches and costing them time. It therefore appears beneficial to provide a transparent system that utilizes the power of such strategies behind the scenes, rather than educate everyone to become an expert user. Such a system could accept NL as input, because it is the most intuitive "query language" and provides more information to the engine than the current paradigm.

Our experiments show that the users were able to make use of their own experience of the engine, their awareness of its idiosyncrasies and/or some trial and error to formulate a better query than a given NL topic, and slightly improve on precision. Yet they still favored JASE's results in many cases, which emphasizes an automaton's advantage of redundant and monotonous computation, and use of an engine's syntactic operators and filters.

We believe our results indicate that NL search systems such as JASE can have practical use in society, and that the NL paradigm can be used to improve the precision of web search.

References

Abney, S., Collins, M. & Singhal, A. (2000), Answer extraction, *in* 'Proceedings of the Sixth Applied Natural Language Processing Conference', Morgan Kaufmann, pp. 296–301.

Agichtein, E., Lawrence, S. & Gravano, L. (2001), Learning search engine specific query transformations for question answering, *in* 'World Wide Web', pp. 169–178.

Brill, E. (1992), A simple rule-based part of speech tagger, *in* 'Proceedings of the Third Conference on Applied Natural Language Processing'.

Brill, E. (1995), 'Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging', *Computational Linguistics* **21**(5), 543–565.

Charniak, E. (1994), Statistical language learning, *in* 'Language and Computers 12', The MIT Press.

Charniak, E. (1997), 'Statistical techniques for natural language parsing', *AI Magazine* **18**(4), 33–44.

Collins, M. J. (1996), A new statistical parser based on bigram lexical dependencies, *in* 'Proceedings of the 34th conference on Association for Computational Linguistics', Morgan Kaufmann, pp. 184–191.

Flank, S. (1998), A layered approach to nlp-based information retrieval, *in* 'Proceedings of the 36th ACL and 17th COLING', Morgan Kaufmann, pp. 397–403.

Frank, E., Paynter, G., Witten, I., Gutwin, C. & Nevill-Manning, C. (1999), Domain-specific keyphrase extraction, *in* 'Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence', Morgan Kaufmann, pp. 668–673.

Jones, K. S. (1972), 'A statistical interpretation of term specificity and its application to retrieval', *Journal of Documentation* **28**(1), 11–21.

Katz, B. (1991), 'Text processing with the start natural language system', *Text, ConText, and HyperText: writing with and for the computer* pp. 55–76.

Katz, B. & Lin, J. (2000), Rextor: A system for generating relations from natural language, *in* 'Proceedings of the ACL 2000 Workshop on Natural Language Processing and Information Retrieval'.

Kupiec, J. (1993), Murax: a robust linguistic approach for question answering using an on-line encyclopedia, *in* 'Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval', ACM Press, pp. 181–190.

Kwok, C., Etzioni, O. & Weld, D. (2001), Scaling question answering to the web, *in* 'World Wide Web', pp. 150–161.

Luhn, H. P. (1957), 'A statistical approach to mechanized encoding and searching of literary information', IBM Journal of Research and Development, 4(4), 600-605.

Munoz, A. (1996), Compound key word generation from document databases using a hierarchical clustering ART model. IDA, Amsterdam.

Page, L., Brin, S., Motwani, R. & Winograd, T. (1998), 'The pagerank citation ranking: Bringing order to the web', Stanford Digital Library Technologies Project.

Porter, M. (1980), An algorithm for suffix stripping, *in* 'Program', Vol. 14, pp. 130–137.

Prager, J., Brown, E., Coden, A. & Radev, D. (2000), Question-answering by predictive annotation, *in* 'Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval', ACM Press, pp. 184–191.

Ravichandran, D. & Hovy, E. (2002), Learning surface text patterns for a question answering system, *in* 'Association for Computational Linguistics Conference'.

Salton, G. (1989), *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*, Addison-Wesley Longman Publishing Co., Inc.

Salton, G. & Buckley, C. (1988), 'Term-weighting approaches in automatic text retrieval', *Information Processing and Management* **24**(5), 513–523.

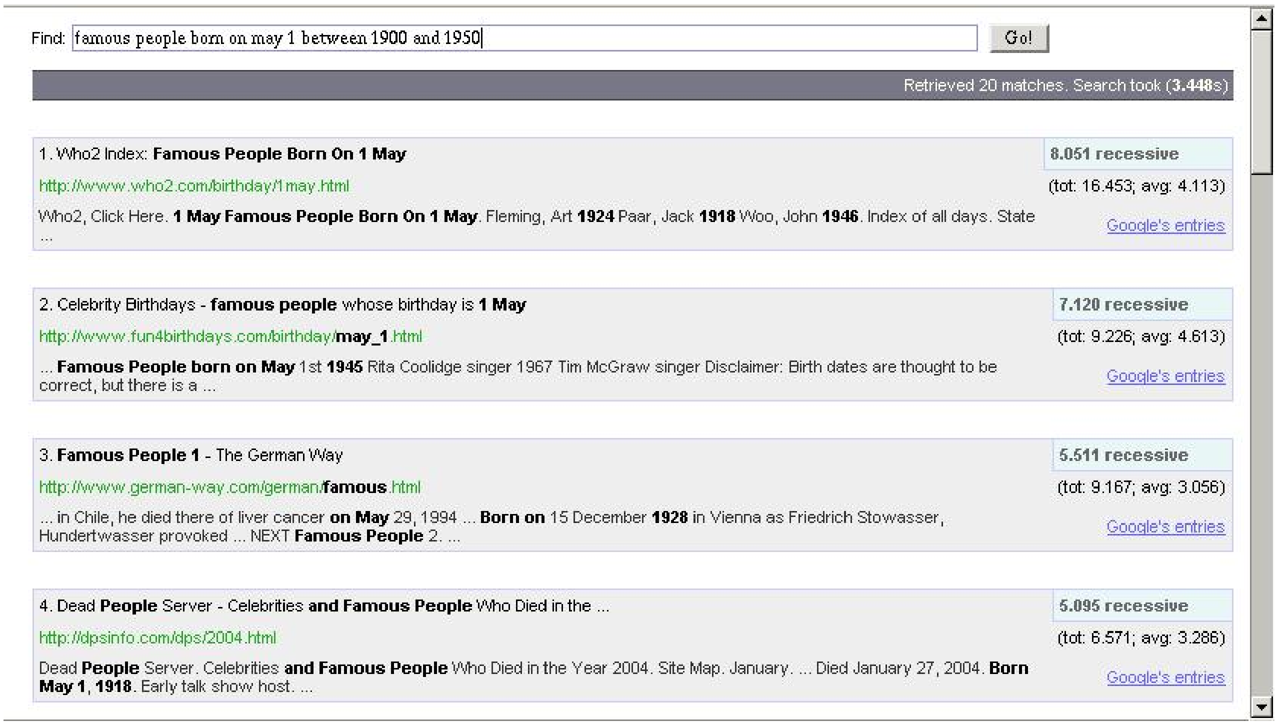
Turney, P. (1999), 'Learning to extract keyphrases from text', Technical Report ERB-1057, National Research Council, Institute for Information Technology.

Turney, P. (2000), 'Learning algorithms for keyphrase extraction', *Information Retrieval* **2**(4), 303–336.

8.1 Appendix: Sample

Figure 2 is a screenshot of our prototype for q13, with the corresponding Google results in Figure 3. Both images date to the time of our evaluation (Oct 2004).

In June 2005, we revisited the top Google hits for this query. The top 10 hits were different to before, perhaps due to PageRank fluctuations and new documents being introduced. JASE's results were, in turn, equally affected. We inspected each of Google's top 20 hits and decided that 3 were relevant, but only one of them appeared in the top 10. JASE managed to extract 8 relevant documents, of which 6 were in its top 10. We were able to identify four times as many "famous people" (to answer the query) from JASE's top 10 hits than from Google's top 20 hits.



famous people born on may 1 between 1900 and 1950

Retrieved 20 matches. Search took (**3.448**s)

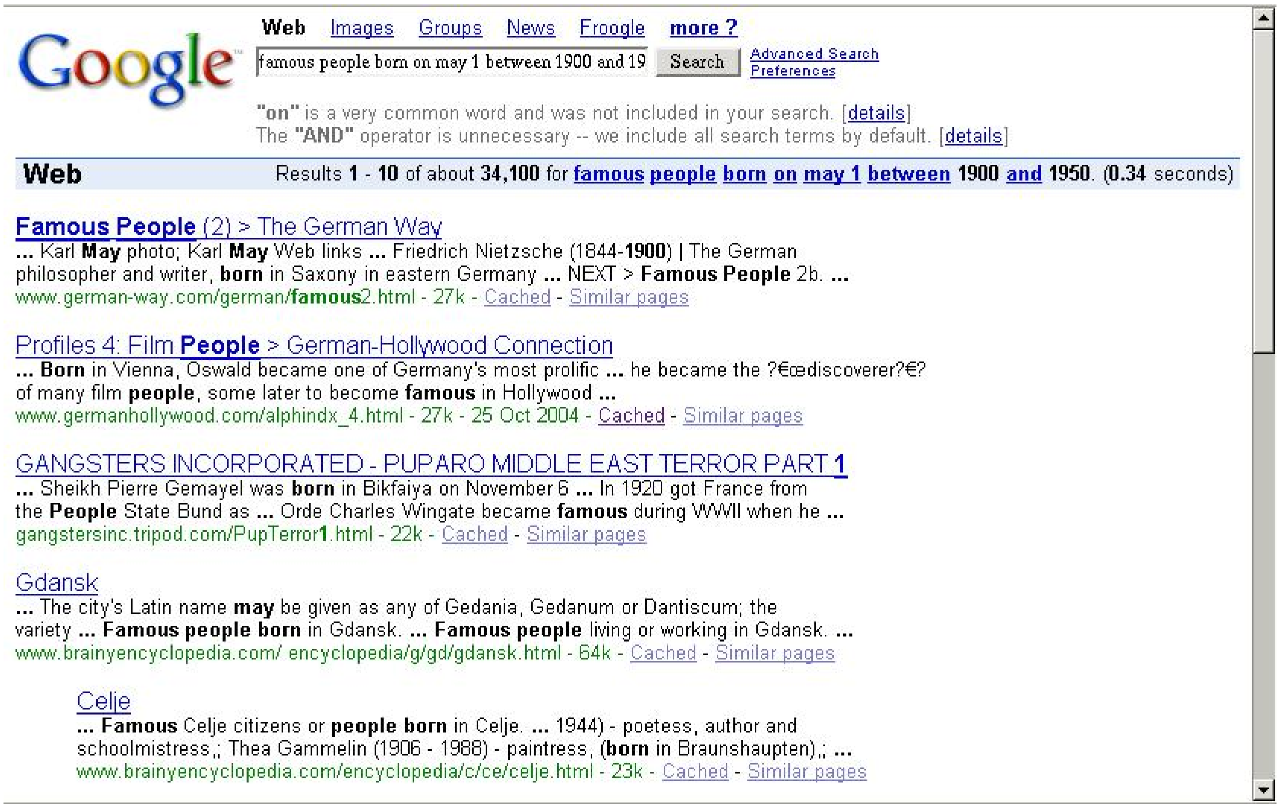
1. Who2 Index: **Famous People Born On 1 May** — **8.051 recessive**
http://www.who2.com/birthday/1may.html — (tot: 16.453; avg: 4.113)
Who2, Click Here. **1 May Famous People Born On 1 May**. Fleming, Art **1924** Paar, Jack **1918** Woo, John **1946**. Index of all days. State ... — Google's entries

2. Celebrity Birthdays - **famous people** whose birthday is **1 May** — **7.120 recessive**
http://www.fun4birthdays.com/birthday/**may_1**.html — (tot: 9.226; avg: 4.613)
... **Famous People born on May** 1st **1945** Rita Coolidge singer 1967 Tim McGraw singer Disclaimer: Birth dates are thought to be correct, but there is a ... — Google's entries

3. **Famous People 1** - The German Way — **5.511 recessive**
http://www.german-way.com/german/**famous**.html — (tot: 9.167; avg: 3.056)
... in Chile, he died there of liver cancer **on May** 29, 1994 ... **Born on** 15 December **1928** in Vienna as Friedrich Stowasser, Hundertwasser provoked ... NEXT **Famous People** 2. ... — Google's entries

4. Dead **People** Server - Celebrities **and Famous People** Who Died in the ... — **5.095 recessive**
http://dpsinfo.com/dps/2004.html — (tot: 6.571; avg: 3.286)
Dead **People** Server. Celebrities **and Famous People** Who Died in the Year 2004. Site Map. January. ... Died January 27, 2004. **Born May 1, 1918**. Early talk show host. ... — Google's entries

Figure 2: JASE's top results for q13



famous people born on may 1 between 1900 and 19

"**on**" is a very common word and was not included in your search. [details]
The "**AND**" operator is unnecessary -- we include all search terms by default. [details]

Web Results **1** - **10** of about **34,100** for **famous people born on may 1 between 1900 and 1950**. (**0.34** seconds)

Famous People (2) > The German Way
... Karl **May** photo; Karl **May** Web links ... Friedrich Nietzsche (1844-**1900**) | The German philosopher and writer, **born** in Saxony in eastern Germany ... NEXT > **Famous People** 2b. ...
www.german-way.com/german/**famous**2.html - 27k - Cached - Similar pages

Profiles 4: Film **People** > German-Hollywood Connection
... **Born** in Vienna, Oswald became one of Germany's most prolific ... he became the ?€œdiscoverer?€? of many film **people**, some later to become **famous** in Hollywood ...
www.germanhollywood.com/alphindx_4.html - 27k - 25 Oct 2004 - Cached - Similar pages

GANGSTERS INCORPORATED - PUPARO MIDDLE EAST TERROR PART **1**
... Sheikh Pierre Gemayel was **born** in Bikfaiya on November 6 ... In 1920 got France from the **People** State Bund as ... Orde Charles Wingate became **famous** during WWII when he ...
gangstersinc.tripod.com/PupTerror**1**.html - 22k - Cached - Similar pages

Gdansk
... The city's Latin name **may** be given as any of Gedania, Gedanum or Dantiscum; the variety ... **Famous people born** in Gdansk. ... **Famous people** living or working in Gdansk. ...
www.brainyencyclopedia.com/ encyclopedia/g/gd/gdansk.html - 64k - Cached - Similar pages

Celje
... **Famous** Celje citizens or **people born** in Celje. ... 1944) - poetess, author and schoolmistress,; Thea Gammelin (1906 - 1988) - paintress, (**born** in Braunshaupten),; ...
www.brainyencyclopedia.com/encyclopedia/c/ce/celje.html - 23k - Cached - Similar pages

Figure 3: Google's top results for q13

Approximative Filtering of XML Documents in a Publish/Subscribe System

Annika Hinze[1] **Yann Michel**[2] **Torsten Schlieder**

[1]University of Waikato, New Zealand
[2]Freie Universitaet Berlin, Germany
a.hinze@cs.waikato.ac.nz
ymichel@inf.fu-berlin.de
torsten.schlieder@gmx.net

Abstract

Publish/subscribe systems filter published documents and inform their subscribers about documents matching their interests. Recent systems have focussed on documents or messages sent in XML format. Subscribers have to be familiar with the underlying XML format to create meaningful subscriptions. A service might support several providers with slightly differing formats, e.g., several publishers of books. This makes the definition of a successful subscription almost impossible. This paper proposes the use of an approximative language for subscriptions. We introduce the design of our ApproXFilter algorithm for approximative filtering in a publish/subscribe system. We present the results of our performance analysis of a prototypical implementation.

1 Introduction

The recent years have seen a new generation of applications based on the principle of *publish/subscribe* (pub/sub): distribution of stock quotes, news articles, or library alerts. A publish/subscribe system is a (distributed) middleware implementing the event-based communication paradigm: A source or *publisher* publishes *event messages* that announce the occurrence of events, i.e., the occurrence of something of interest within the system. Examples are the publication of a new book or CD. *Subscribers* can subscribe to events that are of interest to them; these subscriptions are called *profiles.* The system filters the incoming messages according to the profiles and forwards matched messages to their subscribers.

Publish/susbscribe systems have their origin in alerting services for digital libraries (Salton 1968). In the first generation of alerting systems, event messages contained the full text of documents, such as a newly published scientific paper (e.g., in SIFT (Yan & García-Molina 1995)). A profile would equal a simple Information Retrieval (IR) query using keywords. Note that the concept of filtering documents against a set of profiles has been explored earlier in *information filtering* by the Information Retrieval community. However, the focus there is on information quality, whereas we are looking at efficiency for large scale settings with high numbers of profiles. The focus of publish/subscribe systems lies more on the efficient filtering of structured data sets. Thus, earlier pblish/subscribe systems supported either attribute-value pairs (e.g., in Siena (Carzaniga 1998)) or SQL-like queries (e.g., in CQ (Liu, Pu & Tang 1999)).

Recently, XML-based messages or documents have been used to encode the event messages (e.g., in NiagaraCQ (Chen, DeWitt, Tian & Wang 2000), XFilter (Altinel & Franklin 2000)). Applications are eBusinesses such as online catalogs or digital libraries. Here, a profile is a XML query expressed in XML-QL (NiagaraCQ) or Xpath (XFilter); the definition of which is a rather demanding task for a user who is not familiar with XML query languages. In addition, almost all existing systems assume that the users are well informed about the structure of the event messages and that they are therefore able to create meaningful profiles.

The task of creating a meaningful profile is even more demanding if the system supports different providers of information, e.g., different publishers of music CDs or books, which may use slightly differing catalogue structures. Currently, no system supports filtering over varying structures. In addition, current filter mechanisms detect only documents that contain the exact values a subscription specifies, but it is not possible to detect documents that contain synonymous values.

Typical solutions for this kind of searches in digital libraries are extensions or replacements of search terms with synonyms using a thesaurus or a dictionary (e.g., in the DejaVu system (Gordon & Domeshek 1998)). Other techniques that have been used to explore semantic relationships between terms include user feedback and enriched search interfaces (Rao, Pedersen, Hearst, Mackinlay, Card, Masinter, Halvorsen & Robertson 1995). For substructured data, the problem of approximative results has been extensively addressed for XML search queries (e.g., in (Schlieder 2003, Theobald & Weikum 2002)).

For publish/subscribe systems, the problem of how to extend the profiles and how to efficiently filter using approximations remains open. Note that the issue of how to create thesauri or cost-enriched term lists remains the same problem as for searching. We see this as a separate problem that is not addressed. In this paper, we focus on an efficient filter algorithm for approximate publish/subscribe. We show later that also for algorithms, inspiration may be found in IR solutions, but it is not possible to simply copy these algorithms.

In this paper, we propose an approximative filtering algorithm ApproXFilter to address the problem of approximate filtering. The main challenge for filter algorithms in a publish/subscribe context is efficient filtering of large numbers of profiles. We introduce two forms of an approximative algorithms for filtering XML documents: a time optimized version and a space optimized version. We present a performance analysis of our prototypical implementation and show the usefulness of our approach.

This paper is structured as follows: Section 2 introduces an example scenario that is used to illustrate the concepts throughout the paper and discusses related approaches. In Section 3, we propose the design of an approximative filter and illustrate the design by example. Two implementation variants are introduced in Section 4. We give details about our prototypical implementation in Section 5. Section 6 presents and discusses the results of our analysis of the algorithms and shows the usefulness of our approach. In Section 7, we discuss complementary approaches. The final section summarizes the contributions and indicates future work.

2 Motivation

This section introduces an illustrative example scenario. We show that existing approaches in publish/subscribe systems are not sufficient and discuss related approaches from information retrieval. We explain why the principles of approximative IR cannot be simply copied for filtering.

Assume an online warehouse offers a publish/subscribe mechanism for its books. A user may know in advance that author Smith will publish a book in the near future. But unfortunately, nothing about the final title or other information is known other than it deals with XML.

Publish/subscribe systems supporting keyword subscriptions (e.g., SIFT) would notify about all documents that contain at least one of the values “XML” and “Smith”. The user cannot specify that she prefers books with the title “XML” over books containing a chapter title “XML”. Similarly, the user cannot prefer the author Smith over the editor Smith. Current systems supporting structured XML queries (e.g., XFilter) would result in the contrary: Only exactly matching documents are considered. The XPath query

$$/catalog/book[title = \text{“}XML\text{”}\ and\, author = \text{“}Smith\text{”}]$$

will neither allow for books with a chapter title “XML” nor books of the category “XML” nor books edited by “Smith”, nor other media formats than books (e.g., articles or tutorials) with the appropriate information.

Of course, the user can create a subscription that exactly matches the cases mentioned, but she must know that similar results may exist and how they are represented. Since all results of her expanded query are treated equally, the user still cannot express her preferences. It is important to note that different to a search query, a user of a publish/subscribe system cannot simply reformulate their subscription query until it gives the desired results - false negatives will occur and the subscriber misses information without being aware of it.

For search engines, solution have been proposed to cope with the approximative searches. For example, ApproXQL (Schlieder 2003) is an approximative filter language with corresponding search algorithm. While common query languages will only match on exact values that were requested, ApproXQL also supports the matching on similar values or structures. This is achieved by skipping or rewriting parts of the query using synonyms. ApproXQL supports hierarchical, Boolean-connected query patterns. The interpretation of ApproXQL queries is founded on cost-based query transformations: The total cost of a sequence of transformations measures the similarity between a query and the data and is used to rank the results. All results of an ApproXQL query can be computed in polynomial time with respect to the database size.

Here, we will follow the concept of ApproXQL and re-use the syntax of its language for filtering purposes. Similar to the case of searching, we follow the approach of using cost-based query transformation. For publish/subscribe systems, we have to develop a new filter algorithm; it is not possible to use the underlying approximative search algorithm: The concept of filtering is the reverse to the concept of searching. In searching, a set of documents forms the foundation; they are indexed and the incoming search query is compared to the index keys. In filtering, a set of subscription queries exists; they are indexed and the incoming document is compared to the indexed query keys. Similarly, the concept of ranking does not have an exact equivalent in filtering. For filtering, the documents are sent to the user or not.

3 The ApproXFilter Algorithm

This section describes the principle of the ApproXFilter algorithm. We start by describing the concept of the algorithm and then move on to discuss each of its steps.

ApproXFilter supports the matching of similar values or structures in addition to direct matches. This is achieved by profile query transformations using skipping, inserting, or renaming parts of the query using synonyms. Whenever a profile query is rewritten for a certain document, each of these transformations may create costs. We introduce the concept of *costs* to judge the quality of a document regarding a given query. A cost of zero means highest quality, i.e., the document exactly matches the profile query as defined by the subscriber. The greater the costs, the lower the matching quality of a document.

Document filtering may be seen as a comparison of the document tree to the set of profile query trees (which are combined in a single directed acyclic graph (DAG)). The more similar a document tree is to a given profile tree, the better the match. That is, the better the match the lower the costs. If all possible transformations are supported for a query, each document will match. The costs describe the amount of transformation necessary to reach that match (similar to relevance in Information Retrieval). If only selected transformations are allowed, not all documents will match a given profile. The costs can be seen as a (reverse) measure for the similarity between the documents and the matched profiles. For profiles that are not matched using transformations, and for profiles that are matched creating high costs, the similarity between the profiles and the document is low.

We now introduce the overall structure of the algorithm. Subsequently, we illustrate the algorithm by using our example scenario.

Step 1 - Normalization: After the definition of the subscriptions, transform all ApproXFilter subscriptions into their conjunctive normal form (i.e., Boolean disjunctions combined by conjunctions)

Step 2 - Profile Extension: Extend all subscriptions using the allowed predefined transformations (renaming, skipping, insertion)

Step 3 - Tree-building: Build a subscription match DAG containing all extended subscriptions

Step 4 - Filtering: For each incoming document: Go sequentially through document; concurrently traverse the match DAG depth-first; whenever moving upwards in the match DAG accumulate the costs

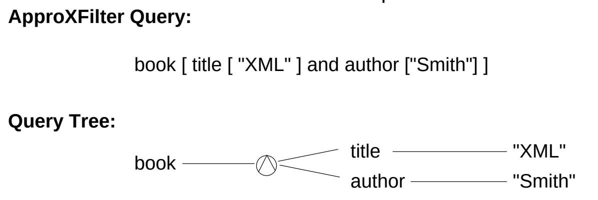


Figure 1: ApproxFilter sample profile query and its query tree (Query 1)

Step 5 - Notification: If the accumulated costs for the matching document are less than a predefined threshold, inform the subscriber about the document

We will now illustrate these steps by applying the algorithm to our example scenario introduced in Section 2. We will use two example subscriptions and one incoming XML document to show the principle of the filtering algorithm.

Normalization Consider the warehouse's publish/subscribe service from the previous section: Our user is still interested in works about XML by author Smith. Based on her interest, she builds the following subscription written in ApproXFilter:

$$Query\,1: book[title[''XML''] \\ and\,author[''Smith'']]$$

The subscription query and its query tree representation are shown in Figure 1. Another user is interested in all database books that also consider XML, are published in 2005. He defines the following query:

$$Query\,2: book[title[''DB''\;and\,''XML''] \\ and\,year[''2005'']]$$

ApproxFilter's syntax is introduced in detail in Section 4. Note that in this paper, we refer to "XML", "Smith", and "2005" as *values* and to 'book', 'title', 'year' , and 'author' as *structures*; both structures and values are referred to as *terms* in a subscription query. Both subscription queries are already normalized.

Profile Extension Using ApproXQL, it is possible to define synonyms or renamings, deletions or skippings, and insertions. For example, the administrators of the warehouse's system may have defined sets of possible transformations for queries regarding print media. In addition, experienced users may define possible transformations. For simplicity, we use very basic transformations as given in Table 1 for profile extensions in our example Query 1.

Method	Changes	Costs
Rename	book → article	4
	title → abstract	4
	"XML" → "RDF"	7
Skip	title	10
	"XML"	20
Insert	*optional*	0

Table 1: Example profile transformations for Query 1

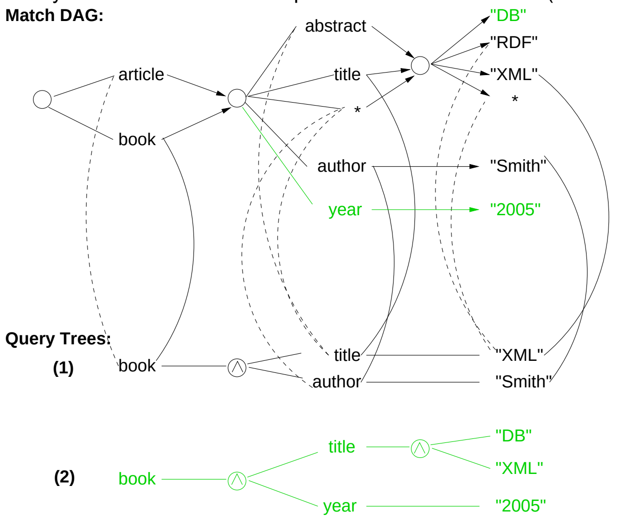


Figure 2: Concept of Match DAG and original query tree. Solid lines for normal edges ($cost = 0$) and dashed lines for additional edges ($cost > 0$)

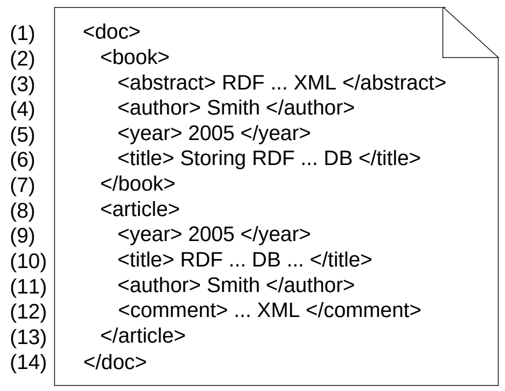
Tree-building The match graph is built as a directed acyclic graph (DAG) for the user profiles as shown in Figure 2. For simplicity, the mapping between the match DAG and the queries is shown only for Query 1; all data regarding Query 2 is shown in a lighter colour. Every term (values and structures) in the extended query is interpreted as a graph vertex.

Figure 2 shows the original profile query (at the bottom) which was extended using the transformations from Table 1: Solid lines between query tree and match DAG represent normal edges, i.e., direct copies from the query tree into the match DAG with no additional costs. Dashed lines represent additional edges, i.e., references created by synonymous structures (e.g., article instead of book) or values ("RDF" instead of "XML") as defined in the transformations table (see Table 1). Additional edges might carry additional costs for the filtering, e.g., as defined as 'Insert' in Table 1. Note that the cost values are chosen arbitrarily. We are aware of the implications of choosing costs, either as a requirement for the user/administrator as well as the challenge of automatic cost assignments. Here, we focus on the performance issues of our approach. In our future work, we plan to address the issues of cost functions and quality.

Note the asterisks in the match DAG in Figure 2: these denote possible skippings of vertices, e.g., the structure 'title' or the value "XML" might be skipped in the filtering. By default, any vertex in the match DAG may be skipped except the root. Skipping vertices may also result in additional costs. The costs for transformations may be defined by system administrators (who should be domain experts) or subscribers.

Filtering Event messages passed into the system are assumed to be well-formed XML documents, such as the simple one in Figure 3. Author Smith has named his book "Storing RDF models in Databases", which is a book about XML technology. The word "XML" does not appear in the title. Conventional publish/subscribe systems would not be able to notify about the book. However, ApproXFilter supports approximative matches and can therefore cover this book by using the appropriate synonyms for values and structures.

The filter algorithm parses the document sequentially and traverses the match DAG in depth-first or-



```
(1)   <doc>
(2)     <book>
(3)       <abstract> RDF ... XML </abstract>
(4)       <author> Smith </author>
(5)       <year> 2005 </year>
(6)       <title> Storing RDF ... DB </title>
(7)     </book>
(8)     <article>
(9)       <year> 2005 </year>
(10)      <title> RDF ... DB ... </title>
(11)      <author> Smith </author>
(12)      <comment> ... XML </comment>
(13)    </article>
(14)  </doc>
```

Figure 3: Example document submitted to the publish/subscribe system for filtering

der. Every difference to the original query is scored with additional costs. For simplicity, the costs are not shown in the example DAG but only in Table 1. For each visited node in the match DAG, the corresponding costs are calculated.

The assignment of costs to each filter step and the final computation of the costs is a non-trivial task. Subsequently, we therefore explain the filtering algorithm and its cost assignments in detail using the example document shown in Figure 3 and the two subscriptions defined earlier that have been processed into the match DAG in Figure 2.

The filter starts parsing the example document (see Figure 3) following the XML tree structure. Each found tag is compared to the match DAG (see Figure 2). Recognizing the tag <`book`> in Line 2 it finds the first matching tag in its internal match DAG (introduced to the DAG by Query 1). It also finds the term 'article' in the DAG as possible renaming for 'book' (introduced by Query 2); here we mainly concentrate on the matchings of Query 1. It then finds the tag <`abstract`> and since this is allowed as a renaming of 'title', it follows this route. Note that the renaming costs (4) are not yet added up but noted in the DAG. In the next step, it compares the words "RDF ... XML" of the abstract to the ones specified in the query for title. First, the filter detects a match of "RDF" and notes the additional costs (7) for this level. When continuing comparing the words, the algorithm detects that "XML" matches the same vertex but with no additional cost (0).

A document is successfully parsed if the profile query (using allowed transformation) was completely executed. An unsuccessful document could have, for example, a mismatching root node such as CD instead of book or article in our example. After a document is successfully parsed, its costs are evaluated by ascending the match graph. Whenever two branches meet, i.e., whenever a forest of subgraphs finds a common root, the lowest branch-cost is taken as the cost to be accumulated upwards. Therefore, the algorithm always takes the "best sub-match" to compute the match-quality of the parsed sub-document for a given query.

After finishing the comparison for the abstract and reaching the closing tag <`/abstract`>, the algorithm moves upwards in the DAG, calculating the costs as the sum of insertions, deletions and renamings (i+d+r): On the leaf level (Level 3) it computes the minimum of the costs for "RDF" (0+0+7) and "XML" (0+0+0) as ($min(7;0)$) and decides on the match of "XML". On Level 2 of the DAG, the abstract is now closed and the algorithm moves forward to the next tag in the document.

Next, the two tags <`author`> and <`year`> are processed. They do not add additional costs for Queries 1 or 2 because both terms are matched, respectively. We do not go into detail for these tags but concentrate on the subsequent tag <`title`> in Line 6. The filter algorithm follows the tag <`title`> as requested in the profile and tests the title content. As when filtering the abstract, it computes the costs for the "RDF". The occurrence of "DB" is considered for Query 2, but we will not go into detail for that query. The costs for the leaf level for Query 1 are only the renaming costs for "RDF" $= 0 + 0 + 7$. Moving upwards in the DAG, two branches meet on the next level: 'abstract' and 'title'. Their costs are calculated as the sum of their individual costs and the costs of their children resulting in $0 + 0 + 4 + (0)$ for 'abstract' and in $0 + 0 + 0 + (7)$ for 'title'. The algorithm computes the minimum costs for Level 2 ($min(4; 7)$) and decides on the match of 'abstract' (Level 2) followed by "XML" (Level 3). On detecting the close-tag <`/book`>, the overall costs regarding Query 1 for the book structure in the given document are summarized as 4. The costs for Query 2 are also calculated now.

The XML document in this example contains references to two works, i.e., two events are published. This is not necessarily required but it is allowed. The filter algorithm continues parsing the document, now concentrating on the article (starting in Line 8). By detecting the close-tag <`/article`>, the overall costs regarding Query 1 for the article structure in the given document are calculated as $4 + 7 = 11$ (renaming 'book', renaming "XML").

Notification A threshold should be defined by the subscriber or a domain expert for limiting the costs that are allowed for results regarding a given profile. Let's assume a threshold of 10 for our example. Documents with costs lower than the threshold are then forwarded to the subscriber of the profile. In our case of Query 1, the reference for the book (cost 4) is selected and the reference for the article (cost 11) is discarded. Consequently, subscriber for Query 1 will receive a notification about Smith's book.

4 Technical Design

In this section, we propose two alternative implementations for the ApproXFilter algorithm: a time optimized and a space optimized variant (in Sects. 4.1 and 4.2).

4.1 Time-optimized Algorithm

This variant of the algorithm's implementation aims at minimizing the time for filtering a given document. To optimize query evaluation, a permutation of all possible vertex compositions is created (see Figure 4). This includes composition of missing vertices as well as the full query structure as defined by the user's profile. Any vertex may be missing except the root vertex.

Each block of boxes in the figure represents a hash set. For each level in the graph, several hash sets can exist. Each hash set but the root has at least one incoming solid arrow (e.g., book and article point to the middle hash-set). The origins of these arrows are all on the same level, which we refer to as the 'current level'. So, for the middle hash-set, the current level equals the root level. A hash-set directly below the current level contains all combinations of terms that can be found anywhere below the current level in the match DAG (when starting from the points of origins of the solid arrows). For our example, the middle hash-set has its origins in the root node; from Figure 2

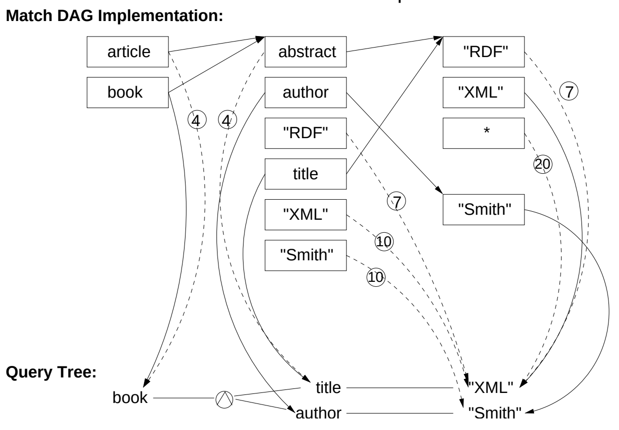


Figure 4: Implementation structure for time-optimized filter algorithm. Solid lines for normal edges ($cost = 0$) and dashed lines for additional edges ($cost > 0$). Dotted lines for skippings of hash sets; Costs greater zero given in circles.

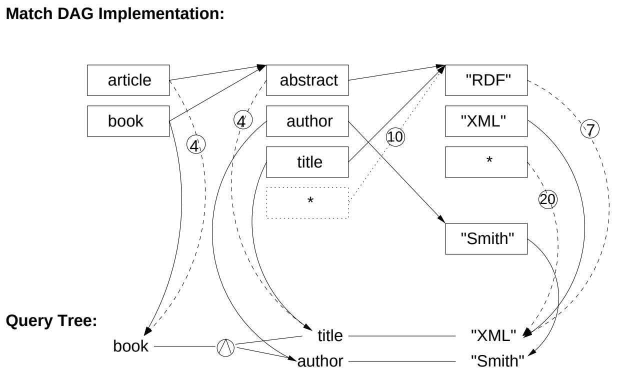


Figure 5: Implementation structure for space-optimized filter algorithm. Solid lines for normal edges ($cost = 0$) and dashed lines for additional edges ($cost > 0$). Dotted lines for skippings of hash sets

we see that below the root node, 6 terms may occur. That means, that the skipping of the title tag has been directly encoded by offering all possibilities of the lower levels also on this level. For this reason, the key "XML" on the middle level carries a cost of 10.

Considering the second level as the current level, below 'abstract' and 'title' in the match DAG, two possible terms can occur ("RDF" or "XML") or the term could be skipped. The skipping has to be made explicit here on the leaf level; the costs for skipping are denoted as 20 as defined.

The dashed arcs in Figure 4 are references to the profile's vertices providing transformation costs whereas the full arcs represent zero costs. Taking our example from above, the arrow pointing from key "RDF" (in the middle hash-set) is annotated with costs for renaming "XML" to "RDF" (7).

Using the structure shown here, the time for evaluating a document is $O(n)$; the space required is $O(n^2)$ where n is the number of vertices in the match DAG as shown in Figure 2. The number of vertices in the DAG could vary considerably depending on the number of profiles p and the number of terms, structures, synonyms. A good estimate would be to assume that n is in the same order of p.

4.2 Space-Optimized Algorithm

This version of the algorithm aims at optimizing space consumption by using smaller data structures. As in the time optimized version, we use hashes to represent the extended query graph. This time, no redundant node entries are allowed in the structure (see Figure 5). Therefore, each hash key is put into the graph only once and in the exact position for representing the original profile structure. All costs are encoded only once.

To skip nodes, we provide wildcard keys (shown as "*" in the dotted box in Figure 5). These keys must be traversed if no hash value matches (using transitive traversal if necessary). For example, the arrow leaving the lowest key in the middle hash-set (with key "*") and pointing to the right upper hash-set (i.e., the hash set with all possible values in title) is annotated with costs for deleting 'title'. If also the author tag would be allowed for deletion, the arrow would also refer to the hash set with the possible author values.

The filter time for this variant is $O(n^2)$, where n refers to the number of vertices. The space required is $O(n)$.

5 Implementation

This section describes the prototype implementation of ApproXFilter. We briefly sketch the prototype's architecture as well as its modules and internal data structures. In addition, we discuss the ApproXFilter language and explain its use for creating a profile.

5.1 Components

The prototype of ApproXFilter is written in Java. We use Xerxes[1] for parsing XML documents. There are three main modules in our implementation as shown in Figure 6:

Profile Service The profile service receives and parses the user-defined profiles that are incoming via the network. It then creates an internal data structure for storing the incoming profiles. The profile service consists of the *profile server* and the *profile worker*. When a connection to the profile server is established, a new profile worker is started. The profile processing incorporates the following steps: worker initialization, profile parsing, profile extension, profile storage, and worker termination.

The profile worker receives and parses the incoming profiles (see upper part in Figure 6). The profile is added to the profile repository. The profile queries are expressed using ApproXFilter; these are translated into an internal profile representation. The profile server manages the list of allowed transformations and their assigned costs. Out of profiles and transformations, the profile server creates the profile match DAG for filtering the profiles.

Document Service The document server receives and parses XML documents; it filters them according to the users' profiles. If profiles match, the profile owners are notified. The central document server dispatches the incoming documents to (distributed) worker threads. The server process is responsible for establishing the connection and passing the work to a dedicated thread.

We regard the matching data structure of profiles as relatively static[2]. Therefore, every document worker obtains a local copy of the global data structure. This copy is only updated when the global profile match DAG changes, i.e, whenever

[1]`http://xml.apache.org/xerces2-j`

[2]This is a viable assumption, e.g., for digital libraries where user profiles describe more long-lived user interests, such as research topics and colleagues.

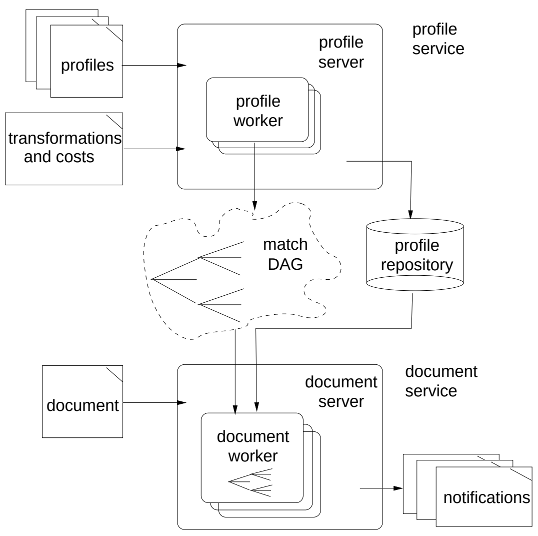


Figure 6: Components of the ApproXFilter engine and their interactions for a set of profiles and a single incoming XML document

the local time-stamp of the match DAG differs from the global time-stamp due to changes by any profile workers. The event processing incorporates the following steps: worker initialization, document parsing, profile evaluation, and worker recycling.

While traversing the incoming XML document, the local data structure is updated with the found vertices and values. The costs of the detected vertices are calculated using local copies of all profiles. To reduce the performance load for updating or initialization, we use time-stamps to detect if the vertex was matched during the current process. If so, we recalculate the costs for this vertex, i.e., we only update the vertex if the new costs are less than the current ones. At last, the complete document costs are calculated by summarizing the costs of all vertices processed in this sequence, i.e., affected by the current document. If a requested vertex was not found in the current sequence, additional costs are added. Additional costs are calculated for insertions as required (i.e., for vertices found in the document that are not mentioned in the profiles). After filtering the document and calculating the costs of the document for all profiles, the costs are compared to the thresholds set for the profiles. Notifications are sent to those subscribers where the document costs are lower than the profile threshold.

Internal Data Structures Effective internal data structures are important for efficient filtering. As seen in Figure 6, a number of internal data structures are held: compact profile trees (bottom), a match structure for filtering document structures (left), and a content-synonym set (top). For the structural matches, we implemented a simplified version of the space optimized DAG; see Figure 7. For the value synonyms (e.g., “RDF” instead of “XML”) we use an additional content-synonym set. For simplicity, in this proof-of-concept implementation we support stricter filtering than the two versions introduced in Section 4 (i.e., fewer skippings). Consequently, the algorithm is more efficient.

For the match-DAG, we maintain a list of all vertices and their synonyms and a compact profile tree structure (see left and bottom in Figure 7).

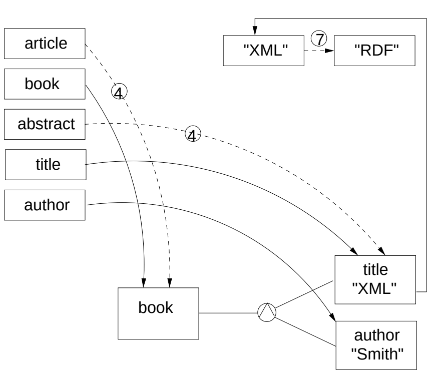


Figure 7: Implemented data structure for matching profile queries; top: value renamings, left: structural renamings, bottom: profile

Each of the vertices refers to all respective profile vertices. For example, ‘article’ and ‘book’ both refer to the profile term ‘book’. This structure facilitates an efficient document parsing process. In addition, every profile-vertex can automatically detect whether it was filtered via the original path or via a transformed one. The latter case results in additional costs.

The profile vertices (bottom of Figure 7) store the profile-defined values within the same vertex (e.g., ‘title’ and “XML” together), and not in a separate vertex as initially proposed in Section 4.2. This merging of content vertices with their parent structural vertices prevents false positives. In our example, the profile would otherwise also match documents containing “Smith” in arbitrary vertices and not only in the ones specified directly in profiles and by allowed transformations. Renamings of values are supported by using the additional content-synonym set (shown in the upper part of Figure 7).

The implemented data structure requires less space than the structure for the space-optimized algorithm version (due to more densely stored profiles); and it’s performance is between the performance of the space-optimized and the time-optimized version. The performance is $O(m^2 + p)$ and the space requirement is $O(m+p)$ where m is the number of structural vertices in the match DAG (i.e., structures ad their synonyms) and p is the number of value vertices in the match DAG (i.e., values and their synonyms).

Element	Content
query	lexpr
expr	lexpr (AND lexpr)* - content (AND content)*
lexpr	label LPAREN expr RPAREN
label	LNAME
content	LITERAL
LPAREN	[
RPAREN	]
LNAME	(‘a’..‘z’-‘A’..‘Z’) (‘a’..‘z’-‘A’..‘Z’-‘_’-‘0’..‘9’)
LITERAL	‘ ” ’ (∼ ‘ ” ’)* ‘”’

Table 2: ApproXFilter profile definition language

5.2 ApproXFilter Language

As already described, we use a subset of the ApproXQL query language for expressing subscriptions in ApproXFilter. Our profile language defines a tree-shaped query string. In our current implementation, we only support conjunctive expressions. The language components used in our implementation are shown in Table 2 as the abstract syntax tree that we used for creating the profile parser using ANTLR [3].

Every profile query consists at least of a labelled expression, "lexpr", having a expression "expr", which is a "content" element. Labels define structural filters, where the label name may consists of any combination of alphanumeric values (see LNAME). "Content" refers to value filters, where a value may be any string enclosed in inverted commas without containing the inverted commas itself (LITERAL).

Translated into our graph profile representation, this describes a single vertex with some content. The query language supports the specification of query strings in which at least one vertex's content-element has to be specified, whereas the parent vertices may be described as simple containers. That is, at least one value filter has to be defined; an arbitrary number of structural filters is allowed.

6 Evaluation of ApproXFilter

In this section, we present the results of the evaluation of our implementation of the ApproXFilter algorithm. We performed functional and quantitative analyses, which are discussed the next two sections.

It is beyond the scope of this paper to reason about the quality of the filter results using structural and/or term-based synonyms; this would reach far into a discussion of IR methodologies and criteria. Therefore, we like to refer instead to similar work done for approximative querying on XML: the quality of the results is the same, since only the filter direction is changed (documents on profiles vs queries on documents). For an extensive discussion see (Schlieder 2003).

The quantitative tests have been performed on a local installation. For a distributed approach, we refer to the multitude of literature for routing algorithms for publish/subscribe, which could be applied here, such as the profile and event forwarding strategies proposed in (Carzaniga 1998).

6.1 Functional Analysis

The functional analysis evaluated the influence of the use of renamings and skippings on the size of the result set. In the first version of the evaluation, the match-DAG was build using the original profiles as defined by the users. In the second version, the profiles were extended using the mentioned transformations.

We tested with a cost-setting for structural conservation, i.e., structural changes cause higher costs then value changes. The costs for this test were defined as follows: skip structure – 15, skip value – 5, rename – 1, insertions – 0. Note that these values are arbitrarily chosen and variable. The results for the filtering of a selection of 50 test documents (using both test versions) are shown in Figure 8. The document IDs appear on the x-axis; the percentage of matched profiles for each document is shown on the y-axis.

The solid boxes represent the proportion of matched profiles for a certain document without transformations. The patterned boxes show the match benefit due to the use of transformations, i.e.,

[3]http://www.antlr.org/doc/index.html

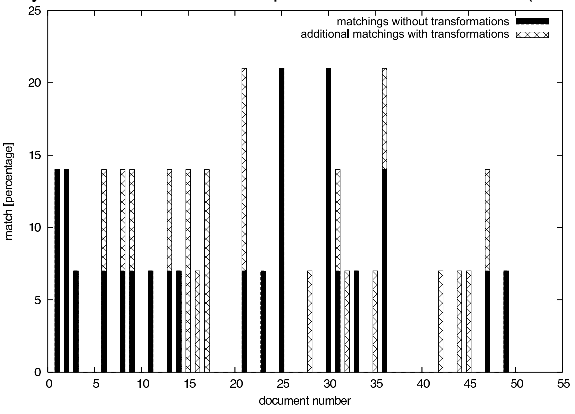


Figure 8: Functional evaluation of the ApproXFilter prototype, matchings with and without transformations

the patterned boxes show the added percentage of matched documents based on transformations. Most documents find more matches after profile transformations. Thus, more users are notified about these documents.[4]

Note that some documents are not matched by any profiles when evaluated strictly, but are matched when approximate matches are allowed (e.g., Documents 15 – 17). These documents originally do not trigger any notifications. On the other hand, for some documents the results are not affected by filter transformations, such as Documents 1 – 3. This means that the similarity between these documents and the profiles was not changed by extending the profiles. Some documents are not matched at all (Documents 37 – 41). For these documents, the similarity between the documents and original profile queries is extremely low, and no similarity is gained by extending the profiles. The algorithms output matched the profile specifications (for details see (Michel & Hinze 2005)). The results of the functional analysis show that the algorithm works as designed: increasing the number of profile matches using approximate filtering.

6.2 Quantitative Analysis

The quantitative analysis evaluates the influence of varying profile numbers on the performance and the space requirements of the algorithm. We present here initial results from a series of tests run on ApproXFilter. This information will assist comparison of later implementations of approximative XML filtering engines.

The test setting used here was similar to that in the qualitative test as described above. For every set of profiles tested, 1000 unique documents were created and filtered. Figure 9 shows both the space usage and performance of our implemented prototype. The left hand side of the figure shows a scale for the time and the right hand side a scale for the space. As argued in Section 5, the space requirement directly depends on the number of vertices in the match DAG. For our test setting, that means that it directly depends on the number of profiles.

For each profile set, we show the mean value for the filter time for one document. The maximum and minimum values indicated show the variation between documents. Note that the variations are stronger for small profile sets. This is due to the stronger influence of single terms on the the filter outcome: both

[4]The stepwise pattern in the results is due to the selection of documents and not inherent to the algorithm.

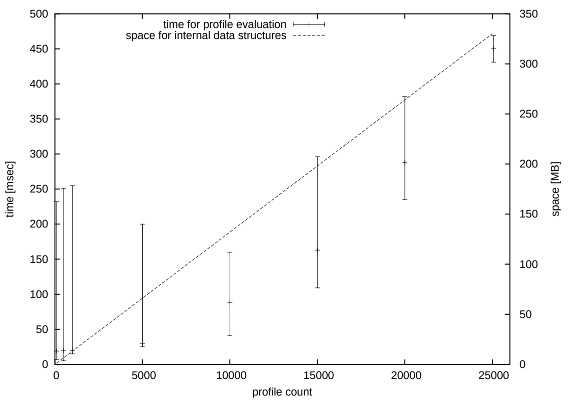


Figure 9: Quantitative evaluation of the ApproXFilter prototype, performance and space requirements depending on number of profiles

documents' structures and profile queries interact to determine the time taken for a filter on one document. Larger samples dampen the effect of this variation.

The performance-related results shown in Figure 9 support our theoretical hypothesis that the algorithm's performance is related to the square of the number of structural vertices.

7 Related Work

Research that is directly related to our approach has been discussed in Section 2. In this section, we look at areas of research that are related but complementary to our work. These areas are flexible queries for semi-structured data, information-retrieval extensions for XML query languages, and filter algorithms for XML documents.

The problem of similarity between keyword queries and text documents has been investigated in information retrieval (Baeza-Yates & B.Ribeiro-Neto 1999). We believe, these models cannot be directly applied to XML documents, since they (1) mostly ignore the structure of XML documents and may therefore lower the retrieval precision, and (2) use models based on term distribution that are of little use for data-centric XML documents. For a discussion of these aspects, see (Fuhr, Lalmas, Malik & Szlavik 2005).

As discussed in Section 2, XML query languages incorporate the document structure and are therefore well suited for applications that query and transform XML documents (Bonifati & Ceri 2000). Almost all query languages for XML support regular path expressions, which allow to specify alternative paths through the data graph and to skip certain subgraphs. Although regular path expressions give some additional flexibility, they also require a considerable knowledge about the data. The user must at least know that some subgraphs must be skipped, that alternative paths exist, and how they look like. Consequently, the user needs substantial knowledge of the data structure to formulate queries. XML query languages that support result ranking are XXL (Theobald & Weikum 2002), ELIXIR (Chinenyanga & N.Kushmerick 2002), XIRQL (Fuhr & Großjohann 2000), ApproXQL (Schlieder 2003).

For event notification systems, we distinguish event centered approaches from document-centered approaches. An example for an event-centered system is A-mediAS (Hinze 2003). In document-centered systems, the events are the publication of a new document. Some publish/subscribe systems use XML-encoded the documents, e.g., NiagaraCQ (Chen et al. 2000) and XFilter (Altinel & Franklin 2000). Profiles are expressed using XML query languages such as XML-QL or Xpath. None of these systems supports approximative filtering of XML documents based on similarity measures.

To the best of our knowledge, the only publish/subscribe system addressing approximate matchings is A-ToPSS (Liu & Jacobsen 2002). Its approach is in sharp contrast to our own. A-ToPSS supports approximate matching for attribute-values pairs using probabilistic measures for both documents and profiles. For each attribute, a possibility distribution may be used to express the confidence that the attribute has a given value. This approach is fundamentally different to the one proposed in this paper. We believe it would be of only limited suitability for text-centered structures; the definition for probability distributions for texts is questionable; it would need substantial knowledge and would unnecessarily burden the users. This approach would map particularly poorly onto XML documents, e.g., because structural changes are not supported and the system works on numerical values only.

8 Discussion and Future Work

Recent publish/subscribe systems increasingly focus on documents send in XML format; subscribers to these systems have to be familiar with the underlying XML format to create meaningful subscriptions. In this paper, we proposed the use of an approximative language for subscriptions.

We introduced the design our ApproXFilter algorithm for approximative filtering in a publish/subscribe system. We discussed two implementation variations that optimized the space usage and the filter performance, respectively. We implemented a proof of concept ApproXFilter prototype that we subjected to qualitative and quantitative testing. The results of our analyses have shown the effectiveness of our approach. To the best of our knowledge, no other filter algorithm for approximative filtering of XML documents exists.

Having proven the concept of approximative filtering, we have a number of open challenges to address: The definition of cost values is a non-trivial problem. Although there are only five cost-related parameters in our prototype, the adjustments have to be done very carefully. The importance of a missing term depends on the filter application. Using low structure-costs results in a more content-based filtering, while lowering the value-cost parameters will result in a more structural filter. We plan to explore the use of user relevance feedback to adjust the costs. A similar dependence on the application domain exists for the definition of synonyms. For this, we would like to explore the use of domain ontologies and personalised ontologies.

One of the next steps will be an extension of our prototype to also support disjunctions. We plan to further analyse and refine the proposed algorithms. In the future, we would like to explore how ApproXFilter could be used in the context of digital library software (internally using XML document representations). It would also be worthwhile to explore a combination of ApproXQL with the Lucene search engine[5] for querying XML documents with subsequent ongoing filtering queries using the ApproXFilter algorithm.

[5] `http://jakarta.apache.org/lucene/docs/index.html`

References

Altinel, M. & Franklin, M. (2000), Efficient filtering of XML documents for selective dissemination of information, *in* 'Proceedings of International Conference on Very Large Data Bases (VLDB '00)', Cairo, Egypt.

Baeza-Yates, R. & B.Ribeiro-Neto (1999), *Modern Information Retrieval*, Addison-Wesley.

Bonifati, A. & Ceri, S. (2000), 'Comparative analysis of 5 XML query languages', *SIGMOD Record* **29**(1), 68–79.

Carzaniga, A. (1998), Architectures for an Event Notification Service Scalable to Wide-area Networks, PhD thesis, Politecnico di Milano, Milano, Italy.

Chen, J., DeWitt, D., Tian, F. & Wang, Y. (2000), NiagaraCQ: A scalable continuous query system for internet databases, *in* 'Proceedings of ACM SIGMOD', Dallas, Texas.

Chinenyanga, T. & N.Kushmerick (2002), 'An expressive and efficient language for XML information retrieval', *JASIST* **53**(6), 438–453.

Fuhr, N. & Großjohann, K. (2000), XIRQL: An extension of XQL for Information Retrieval, *in* 'Proceedings of ACM SIGIR Workshop On XML and Information Retrieval', Athens, Greece.

Fuhr, N., Lalmas, M., Malik, S. & Szlavik, Z., eds (2005), *Advances in XML Information Retrieval: Third International Workshop of the Initiative for the Evaluation of XML Retrieval, INEX 2004, Germany, December 6-8, 2004*, Vol. 3493 of *LNCS*.

Gordon, A. S. & Domeshek, E. A. (1998), Deja Vu: a knowledge-rich interface for retrieval in digital libraries, *in* 'Proceedings of 3rd International Conference on Intelligent User Interfaces (IUI '98)', San Francisco, California, United States.

Hinze, A. (2003), A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service, PhD thesis, Freie Universität Berlin.

Liu, H. & Jacobsen, H.-A. (2002), A-topss - a publish/subscribe system supporting approximate matching, *in* 'Proceedings of International Conference on Very Large Data Bases (VLDB'02)', Hong Kong, China.

Liu, L., Pu, C. & Tang, W. (1999), 'Continual queries for internet scale event-driven information delivery', *IEEE Tranctions on Knowledge and Data Engineering* **11**(4), 610–628. Special issue on Web Technologies.

Michel, Y. & Hinze, A. (2005), ApproxFilter - an Approximative XML-based Filter Engine, Technical Report CS-06/2005, University of Waikato, New Zealand.

Rao, R., Pedersen, J. O., Hearst, M. A., Mackinlay, J. D., Card, S. K., Masinter, L., Halvorsen, P.-K. & Robertson, G. G. (1995), 'Rich interaction in the digital library', *Communications of the ACM* **38**(4), 29–39.

Salton, G. (1968), *Automatic Information Organization and Retrieval*, McGraw-Hill, New York.

Schlieder, T. (2003), Fast Similarity Search in XML Data, PhD thesis, Freie Universität Berlin.

Theobald, A. & Weikum, G. (2002), The index-based XXL search engine for querying XML data with relevance ranking, *in* 'Proceedings of Advances in Database Technology (EDBT '2002)', Prague, Czech Republic.

Yan, T. W. & García-Molina, H. (1995), SIFT - a tool for wide-area information dissemination, *in* 'Proceedings of the USENIX'1995', New Orleans, Louisiana, USA.

Manufacturing Opaque Predicates in Distributed Systems for Code Obfuscation

Anirban Majumdar **Clark Thomborson**

Secure Systems Group, Department of Computer Science
The University of Auckland,
Private Bag 92019, Auckland, New Zealand.
Email: {anirban|cthombor}@cs.auckland.ac.nz

Abstract

Code obfuscation is a relatively new technique of software protection and it works by deterring reverse engineering attempts by malicious users of software. The objective of obfuscation is to make the logic embedded in code incomprehensible to automated program analysis tools used by adversaries. Opaque predicates act as tool for obfuscating control flow logic embedded within code. In this position paper, we address the problem of control-flow code obfuscation of processes executing in distributed computing environments by proposing a novel method of combining the open problems of distributed global state detection with a well-known hard combinatorial problem to manufacture opaque predicates. We name this class of new opaque predicates as *distributed opaque predicates.* We demonstrate our approach with an illustration and provide an extensive security analysis of code obfuscated with *distributed opaque predicates.* We show that our class of opaque predicates is capable of withstanding most known forms of automated static analysis attacks and a restricted class of dynamic analysis attack that could be mounted by adversaries.

Keywords: Code obfuscation, opaque predicates, distributed predicate detection, software protection, mobile code protection, and distributed systems security.

1 Introduction

Software obfuscation is a protection technique for making code unintelligible to automated program comprehension and analysis tools. It works by performing semantic preserving transformations such that the difficulty of automatically extracting computational logic out of the code is increased. The first formal definition of obfuscation was given by Barak *et al.* (2001) where an obfuscator was defined in terms of a compiler that takes a program as input and produces an obfuscated program as output. Two important conditions that need to be preserved while making this transformation are (a) *functionality:* the obfuscated program should have the same functionality (input/output behaviour) as the input program, and (b) *unintelligibility:* the obfuscated program should be unintelligible to the adversary in some sense. Barak *et al.* defined an obfuscation method as a failure if there exists at least one program that cannot be completely obfuscated by this method, that is, if any adversary could learn something from an examination of the obfuscated version of this program that cannot be learned (in roughly the same amount of time) by merely executing this program repeatedly. Their negative result established that every obfuscator will fail to completely obfuscate some programs.

Since Barak's landmark paper on the impossibility of obfuscation, focus has shifted to finding obfuscating transforms that are *difficult* (but not necessarily *impossible*) for an adversary to reverse engineer. The goal of such research is to find sufficiently difficult transforms such that the resources required for undoing them are too expensive to be worth the while of adversaries. Following this line of research, we propose in this contribution, an obfuscation technique derived from the combination of an instance of a hard combinatorial problem and the difficult problem of global state detection in distributed systems.

Depending on the size of software and the complexity of transforms, a human adversary may find the obfuscated code difficult to comprehend. However, as Thomborson *et al.* (2004) noted, software that is simple and manageable enough to be completely analysed by human adversaries could presumably be redeveloped from scratch by attackers at reasonable cost. It is up to the software developer to decide against using complicated obfuscation transforms that might overwhelm the performance of his simple efficient software. We will not address issues related to performance/security tradeoffs in this contribution; nevertheless, the purpose of making such observation at the beginning of this paper is to justify the focus of this paper on an obfuscation method that increases the difficulty of analysing complex programs.

Distributed computing obfuscation could be useful in a number of practical scenarios where it is necessary to maintain code confidentiality. In the first example, consider a distributed electronic commerce bidding scenario where the bidders download seller's code for bidding. The seller's code may contain privileged information such as reserve price and prioritized selection list of bidders (such as frequent bidders may have higher rating than first time bidders). The seller would like to keep such information confidential to the bidders, especially when their programs are executing on hosts owned by bidders, at least for the duration of the auction. Code obfuscation would serve as an appropriate tool in achieving this objective. Secondly, consider a grid computing scenario, like the SETI@home (2005) setup, where scientific computation codes are downloaded on untrusted personal computers connected to the global network of loosely-coupled machines. These machines are owned by users willing to contribute a portion of their machine's processing power and time for helping the project compute a section of its scientific result by executing the downloaded code. Here too, it may

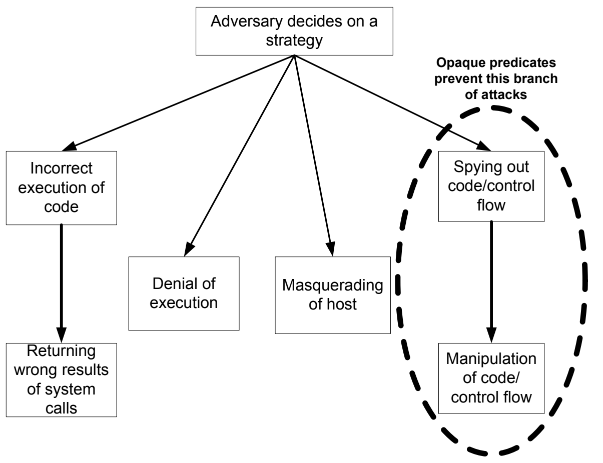

Adversary decides on a strategy
Opaque predicates prevent this branch of attacks
Incorrect execution of code
Denial of execution
Masquerading of host
Spying out code/control flow
Returning wrong results of system calls
Manipulation of code/ control flow

Figure 1: The *attack tree.* The class of attacks marked with the dotted oval are specifically addressed by control-flow obfuscation using opaque predicates.

be desirable that scientific computation logic be kept obscure to the owner of the host. Lastly, distributed obfuscation would be most useful in hiding watermark construction code (Palsberg *et al.* 2000, Nagra & Thomborson 2004) which are used for proving ownership of software. Note that ownership proofs are most important during the economic lifetime of the software product. In all three scenarios, obfuscation need not be perfect in the sense of Barak. Instead, obfuscation is useful if it delays the release of confidential information for a sufficiently long time (Hohl 1998). Secondly, any obfuscation technique would increase the confidence of the code-sender, but might decrease the confidence of a code-executer because it would make it harder to understand what the code is doing.

Control-flow obfuscation by means of opaque predicates was introduced by Collberg *et al.* (1998). An opaque predicate is a construct with true/false outcome. The opaqueness property of predicates is attributed by the fact that though their outcome is known at obfuscation time, it is hard for a deobfuscator to deduce from automated program analysis trace. These constructs are specifically useful for addressing attacks originating out of spying the control-flow as illustrated in the *attack tree* of Figure 1. This branch confidentiality is achieved by obscuring the real control flow of behaviours behind irrelevant statements that do not contribute to the actual computations. An adversary with no semantic understanding of correct control-flow of the code will also find it hard to do purposeful manipulation of the code.

The rest of the paper is structured as follows: In section 2, we introduce notation for discussing distributed opaque predicates. Section 3 illustrates with an example how distributed opaque predicates could be constructed in distributed systems. In section 4, we present a security analysis of our technique. We conclude our paper with a summary and discussion of future work in section 5.

2 The concept of distributed opaque predicates and global states in distributed systems

We define a *distributed opaque predicate* (Φ) as an opaque predicate which depends on local states of multiple processes spread across the distributed system for its evaluation. The activity of each process is modeled as execution of a sequence of events. Communication in distributed systems is accomplished through the communication primitive events *send(m)* and *receive(m)*, where m denotes the message. In asynchronous message-passing systems, information may flow from one event to another either because the two events belong to the same process, and thus may access the same local state, or because the two events are of different processes and they correspond to the exchange of a message.

Without a global clock, events can be ordered only based on the notion of causality which states that two events are constrained to occur in a certain order only if the occurrence of the first may affect the outcome of the second. In distributed systems, we use a happened-before relation, $\rightarrow$ between states to denote this causality (Lamport 1978). The happened-before relation can be formally stated as: $a \rightarrow b$ if and only if: a occurs before b in the same process or the action following a is a send of a message and the action preceding b is a receive of that message. Two states for which the happened-before relation does not hold in either direction are said to be *concurrent.* The concurrency relation $\|$, can be formally stated as: $a \parallel b \Rightarrow (a \not\rightarrow b \wedge b \not\rightarrow a)$. A set of states is called a *consistent cut* if all states are pairwise concurrent.

Palsberg *et al.* (2000) defined the concept of *dynamic* opaque predicates as a possible improvement over *static* opaque predicates defined originally by Collberg *et al.* (1998). Their dynamic opaque predicates were constant over a single program run but varied over different program runs. We extend their concept of dynamic opaque predicates by designing distributed opaque predicates to be *temporally unstable.* A temporally unstable distributed opaque predicate can be evaluated at multiple times at different program points $(t_1, t_2, ...)$ during a single program execution such that the values $(v_1, v_2, ...)$ observed to be taken by this predicate are not identical, that is, there exists i, j such that $v_i \neq v_j$. There are a couple of advantages of making distributed opaque predicates temporally unstable. The first one concerns its reusability; one predicate can be reused multiple times to obfuscate different control flows. The second one relates to its resilience against static analysis attacks. As will be explained later in details, distributed opaque predicate values (v_i) depend on predetermined embedded message communication pattern between different processes participating in maintaining the opaque predicate. The communication pattern serves as an invariant for maintaining the consistency of local states updates and these in turn make the predicate go true or false at desired program locations. It is hard for the attacker to statically deduce predicate values because this pattern is:

- distributed over the processes.
- generated on-the-fly only when processes execute.

Structurally, we design distributed opaque predicates to be relational in nature and of the form:

$$\Phi : [(a + b + c + \ldots + n) \ \Re \ K]$$

where $(a, b, c, \ldots, n)$ are integers whose values are set by individual processes (this forms the local state of the process, as explained in the next section), $\Re$ denotes an equality (inequality) operator such as `'='` (`'!='`) and K is a constant. Opaque predicates that are structurally relational are stealthy in the following sense: an adversary who discovers a relational construct in a program cannot conclude with absolute certainty that it is a distributed opaque predicate since common conditional constructs appearing in programs are often relational in nature. But the most important purpose of making distributed opaque predicates structurally relational lies in the difficulty of detecting this class of predicates in the context of distributed global state monitoring. Relational predicates cannot be written as a Boolean expression of local predicates and therefore presents foremost difficulty in distributed global state detection (Chase & Garg 1995). Our rationale will be further clarified in section 4, where we provide a full security analysis for our class of distributed opaque predicates. A detailed discussion on the difficulty of distributed global state monitoring is outside the scope of this contribution and the reader is encouraged to see Chase & Garg (1995) and the references contained therein.

In the next section, we will illustrate how distributed opaque predicates can be generated from an instance of a hard combinatorial problem in distributed systems. An obfuscator will automatically embed distributed opaque predicates in a distributed systems program and insert *send/receive* primitives for generating a predetermined communication invariant. The communication invariant, in turn, maintains the consistency of local states, that is, the value of each component in the predicate (Φ) so that the predicate holds true (Φ^T) or false (Φ^F) at predetermined control-flows decided by the obfuscator and we will argue that to an attacker, predicate value at every obfuscated control-flow seems unknown ($\Phi^?$).

3 Generation of distributed opaque predicates for distributed systems

We present here different design issues an obfuscator needs to deal with and a step-by-step approach for generating distributed opaque predicates in the context of distributed computing obfuscation.

3.1 Selecting/spawning *guard* processes

Let us assume that a distributed computing system consisting of a set of n inter-communicating processes, denoted by $\{P_1, P_2, P_3, ..., P_n\}$, executes on multiple heterogeneous hosts. Assuming that the control-flow of process P_1 is to be obfuscated using distributed opaque predicates, the obfuscator selects or spawns a certain number of *guard* processes to aid in the obfuscation of P_1. Since processes in distributed systems typically collaborate through message exchanges to achieve a particular task, the set of *guards* could be those processes P_1 frequently communicates with. The actual number of *guards* employed in the obfuscation of a single process may depend dynamically on the availability of processes. However, the obfuscator may spawn dummy processes to serve as *guards* if there are not enough processes in the system to do this task. The basic idea is to distribute the local states formed in the construction of distributed opaque predicate in P_1 amongst the *guards* and embed a communication pattern in the form of *send/receive* calls that will update respective local states of processes to previously known values. The local state update rules and communication pattern embedding are described in the following subsections.

We illustrate the process interaction architecture in Figure 2. For the demonstration to follow, we have selected two processes, P_2 and P_3, to serve as *guards* for P_1. Local state for each process i is denoted by the variable $p_i.v$. P_1 could, in turn, serve as a *guard* process for helping in obfuscating any other process within the system but we have excluded that possibility for the sake of keeping this illustration simple.

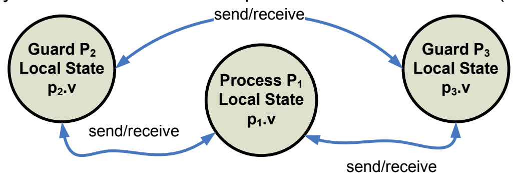

send/receive
Guard P2 Local State p2.v
Process P1 Local State p1.v
Guard P3 Local State p3.v
send/receive
send/receive

Figure 2: The protected process P_1 with local state $p_1.v$ and two *guards* P_2 and P_3 with local states $p_2.v$ and $p_3.v$ respectively.

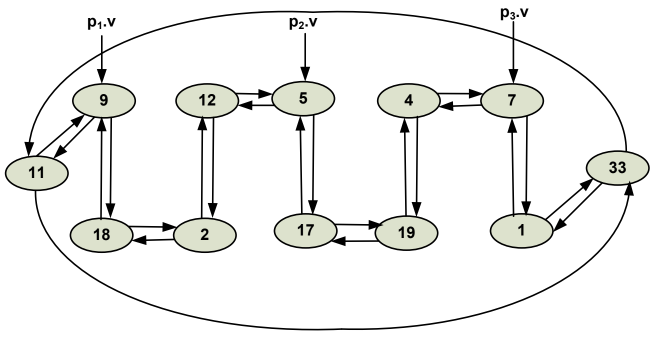


Figure 3: The doubly circular linked-list configurations of P_1, P_2 and P_3 initialised with elements from set S. Each copy of the list is also initialised with an initial pointer location ($p_1.v$, $p_2.v$, or $p_3.v$) respective to the process it is sent.

3.2 Adapting a Knapsack problem instance for distributed computing obfuscation

We now consider an instance of a hard combinatorial problem called Knapsack problem (Garey & Johnson 1979) and show that it can be adapted for manufacturing distributed opaque predicates. The original 0/1-Knapsack problem can be stated as follows:

Given a set $S = \{a_1, a_2, \ldots, a_n\}$ of positive integers and a sum $T = \sum_{i=1}^{n} x_i a_i$ where each $x_i \in \{0, 1\}$, find x_i. This decision problem has been shown to be NP-complete. In adapting this problem for manufacturing distributed opaque predicate, the obfuscator selects the set S of positive integers and x_i's according to some predetermined sum T. An adversary through careful static analysis and reverse engineering may come to learn about set S and sum T. However, given an arbitrarily large set, the hard problem for him is to not only decide if a solution vector x exists but to also determine the vector at precisely the program points $(t_1, t_2, ...)$ where distributed opaque predicates are used to obscure the control-flow of P_1. This is hard since the distributed opaque predicates are constructed from local states (the values range in set S) of *guard* processes and P_1 and local states dynamically change depending on the interaction pattern between processes. This underlying concept will gradually evolve as we describe our methodology.

For our illustration, we select an arbitrary set as:

$$S = \{11, 9, 18, 2, 12, 5, 17, 19, 4, 7, 1, 33\}$$

After dynamically selecting/spawning the *guards*, process P_1 and the *guards* are each initialised by passing a dynamic data structure, such as a doubly circular linked-list, initialised with the elements of the set S. This is illustrated in Figure 3.

In addition to initialising the linked lists with elements of set S, each copy is also initialised with an initial pointer location respective to the process the list is sent. Node values corresponding to the pointer locations form the local state of that particular process. For our illustration with three processes, the list is initialised with three pointers: $p_1.v$ for P_1, $p_2.v$ for P_2, and $p_3.v$ for process P_3. Messages are exchanged between the *guards* $\{P_2, P_3\}$ and P_1 according to an embedded communication pattern. Generation of this predetermined communication pattern will be discussed shortly. Considering an arbitrary sum for our illustration as $T = 27$, the corresponding solution vector x for the sum T is:

$$x = \{0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0\}$$

For our three process illustration, this solution vector corresponds to $p_1.v = 18$, $p_2.v = 5$, and $p_3.v = 4$. The *distributed opaque predicate* (Φ) thus formed in this case would be:

$$\Phi : p_1.v + p_2.v + p_2.v = 27$$

In the following subsections, we will explain how to coordinate the local state update values between processes by controlling their interaction pattern such that Φ is satisfied at precisely the program points where the obfuscator decides. We note that there could be other possible solution vectors for set S and a similar approach for constructing distributed opaque predicates could be used with different sets of *guards* and the same sum T.

3.3 Defining the local state update rules

The local state update rules defined on inter-process message communication events are defined as follows:

`receive(m)` $\Longrightarrow$ `pointer shifted right of the current node`

`send(m)` $\Longrightarrow$ `pointer shifted left of the current node`

Thus, if the local state of P_1, defined by $p_1.v$, is 9 at a certain point in P_1's execution and if P_1 receives a message, the local state will change to 18. Similarly, if P_1 sends a message, the local state changes to 11. Since the distributed opaque predicate is constructed by composing the local states of individual processes and each local state value (corresponding to the pointer location) fluctuates when processes send or receive messages, the predicate will alternate between true/false outcomes throughout the run.

3.4 Selection of communication pattern and message types

As stated earlier, local state update of individual processes take place according to an embedded invariant communication pattern. This predetermined pattern is generated when the embedded *send/receive* calls in processes P_1, P_2, and P_3 get executed. The calls could be embedded by the obfuscator by tracing and annotating the processes with *send/receive* primitives, much in the same way dynamic watermarking algorithms annotate programs for inserting watermark building code (Nagra & Thomborson 2004). However, there are a couple of problems with adopting this approach for embedding the communication pattern. First of all, embedded *send/receive* calls will generate an arbitrary pattern for each run of the program unless they are controlled in some way. Secondly, because of the nondeterminism and latency associated with asynchronous message passing, there is no guarantee of causal delivery of messages. Thus, we have to ensure message exchanges satisfy FIFO (First-in-first-out) delivery. This delivery order ensures for all messages m and m':

$$\mathrm{send_i(m)} \rightarrow \mathrm{send_i(m')} \Rightarrow \mathrm{deliver_j(m)} \rightarrow \mathrm{deliver_j(m')}$$

In distributed systems, the notion of global clock is absent. We propose using vector clocks (Mattern 1989) for solving these two problems. Using vector clock, event orderings based on increasing clock values are guaranteed to be consistent with causal precedence. Before going into a detailed discussion on its usage for constructing the predetermined communication pattern, we briefly provide a general overview of vector clocks.

Vector clock of a system of n processes is an array of n logical clocks, one per process. A local copy of the vector clock is kept in each process P_i, contributing a local state in the construction of distributed opaque predicate. A notation of $VC_i^b[i]$ denotes the logical clock value of P_i at *send/receive* event b. $VC_i^b[j]$ denotes the time $VC_j^a[j]$ of last event a at P_j that is known to have happened before its local event b. The vector clock algorithm update rules could be specified as:

- If a and b are successive events in P_i, then $VC_i^b[i] = VC_i^a[i] + 1$.
- Also, if b denotes *receive(m)* by P_i with a vector timestamp t_m, then $VC_i^b[k] = max\{VC_i^a[k], t_m[k]\}$, for all $k \neq i$.

Three obfuscation-specific message classes are used for message exchanges between process P_1 and the *guards* P_2 and P_3. These message classes help in maintaining consistency of vector clock values and local state updates. Each class is identified by a special tag. The first class is identified by the tag `SYSTEM`. Messages of this class may originate in either P_1, P_2 or P_3 and carry vector timestamp in them. Also, when a process participates in a *send/receive* event of `SYSTEM` type messages, it updates its vector clock and local state according to the state update rules specified in the previous subsection. The second class, `REQUEST`, type message may only originate at P_1 since it is used to request local state values for the *guards*. This class also carries vector timestamp and causes vector clock updates but does not cause any change in local state when received by the *guards*. The third type of message is identified by tag `RESPONSE`. This type is identical to the second class of messages with the exception that these originate at *guards* and are received by P_1. `RESPONSE` messages are used by *guards* to send local state values back to P_1 against incoming `REQUEST` messages.

An example of predetermined communication pattern is illustrated with processes P_1, P_2, and P_3 in the event-time diagram of Figure 4. An event-time diagram maps each event against time and state changes are effected by exchange of messages. The embedded *send/receive* calls in processes P_1, P_2, and P_3 generate this communication pattern. The vector clock value for each participating process is indicated within the square brackets and the value of local state is indicated in variable $p_i.v$, where i denotes the process number. Thick arrows in the figure denote `SYSTEM` type messages. Thin arrows denote `REQUEST` type messages and dashed arrows represent `RESPONSE` type messages.

As evident from Figure 4, asynchrony of message passing induces concurrency within the system. Because of this concurrency, an adversary will find it difficult to monitor local state changes occurring between the processes from outside and determine if distributed opaque predicate (Φ) is satisfied at a particular program location in a particular run. Along the

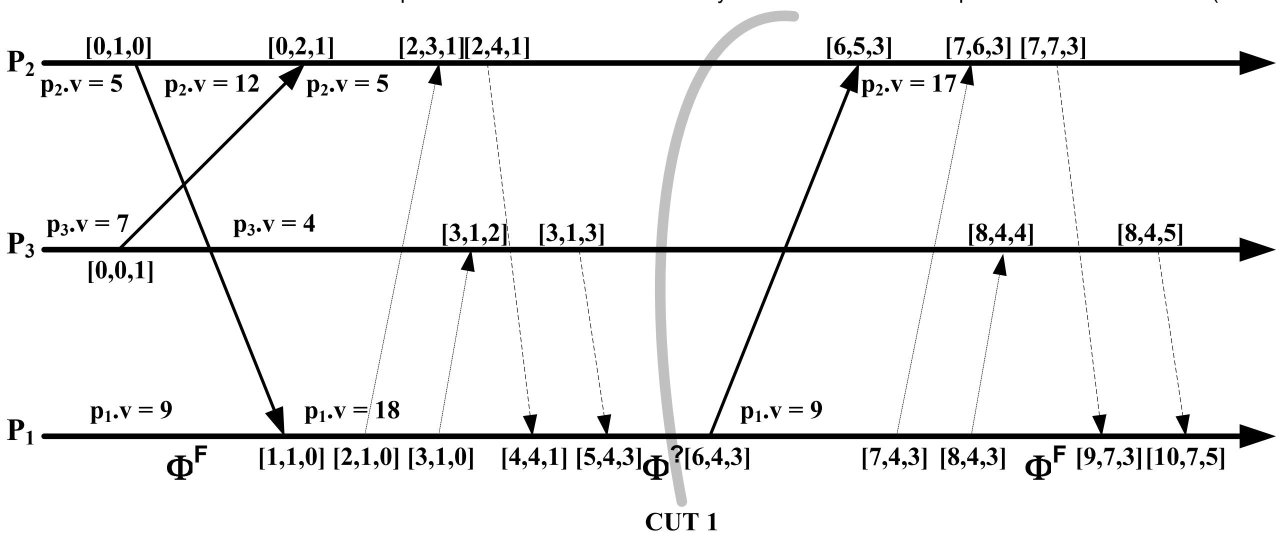


Figure 4: The invariant in the form of a predetermined nondeterministic communication pattern is embedded by the obfuscator into P_1, P_2 and P_3. The update pattern of local states can be traced from Figure 3. Thick arrows denote `SYSTEM` type messages, thin arrows denote `REQUEST` type messages, and dashed arrows denote `RESPONSE` type messages.

timeline of process P_1, we have labeled the value of predicate (Φ) between two successive events distinguished by vector clock values. A (Φ^T) label implies that the predicate is guaranteed to hold true within that event interval (successive events) for that particular run. Similarly, (Φ^F) implies that the predicate is guaranteed to be false within that interval for that particular run. A label denoted by ($\Phi^?$) along the timeline implies that the predicate value is unknown since Φ is not guaranteed to hold.

Moreover, in Figure 4, it seems that (Φ) would be satisfied at `CUT1` since the local states $p_1.v = 18$, $p_2.v = 5$, $p_3.v = 4$ add up to 27. However, note that there is no guarantee of Φ being satisfied at `CUT1`. This guarantee cannot be made because of the following two special cases that could arise out of nondeterminism:

3.4.1 No guarantee on message delivery order

Consider the case from Figure 4 where the message (henceforth referred to as message a) originating from P_3 at vector clock value $[0, 0, 1]$ reaches before the message (henceforth referred to as message b) originating at vector clock value $[0, 1, 0]$ of P_2 is sent by process P_2. This situation is depicted in Figure 5.

When message a reaches *guard* process P_2, the vector clock value changes to $[0, 1, 1]$ and P_2's local state, $p_2.v$, changes from 5 to 17 (refer to Figure 3). *Guard* process P_3's local state, $p_3.v$, changes from 7 to 5. However, after P_2 sends message b at vector clock $[0, 2, 1]$, its local state reverts to 5. When the probe messages (`REQUEST`) are sent by P_1 after the vector clock state $[1, 2, 1]$, the local state values returned from processes P_2 and P_3 are $p_2.v = 5$ and $p_3.v = 4$ respectively. By the time the probe messages are sent to P_2 and P_3, process P_1 has already changed its local state value, $p_1.v$, to 18. The local state values of processes P_1, P_2, and P_3 add up to 27 and the distributed opaque predicate (Φ) is satisfied at `CUT1`.

3.4.2 No guarantee on message delivery

Now consider the case where after the first receive of message b at $[1, 1, 0]$ by process P_1, it cannot be

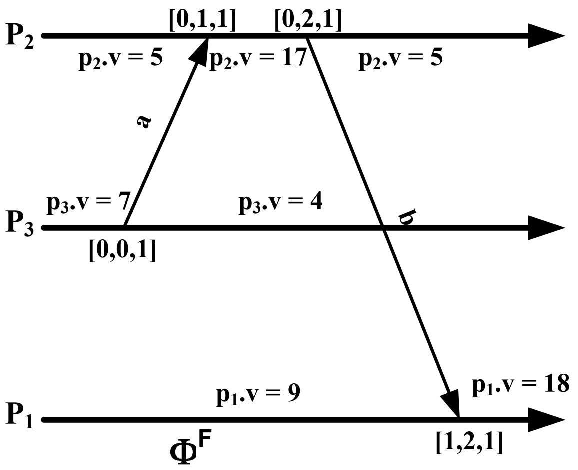


Figure 5: No guarantee on message delivery order. Messages a and b are swapped and (Φ) is satisfied at `CUT1`.

guaranteed that the message a from *guard* process P_3 originating at $[0, 0, 1]$ has reached *guard* process P_2. This guarantee cannot be made because of the nondeterministic nature of asynchronous message-passing. This situation is depicted in Figure 6.

As seen from the figure, since *guard* process P_2 changes its local state to $p_2.v = 12$, the local state values of processes P_1, P_2, and P_3 do not add up to 27. Consequently, the distributed opaque predicate (Φ) is not satisfied at `CUT1`. In yet another specialization of this case, message b may reach process P_1 even before message a originates from *guard* process P_3. In this case, *guard* process P_3 will maintain its local state value at $p_3.v = 7$. Thus, the predicate value will also not be satisfied in this case since the sum of the local state values of processes does not add up to 27.

Thus, we can generally observe, from the nondeterministic communication pattern of Figure 4, that while designing the communication invariant, crossover message-passing patterns will cause nondeterminism within the system and this property could be utilized by the obfuscator to confuse attackers into falsely believing that a distributed opaque predicate will be guaranteed to hold true or false at a particular

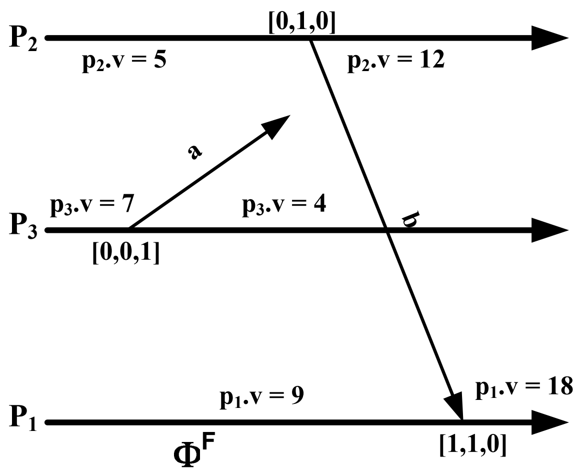


Figure 6: No guarantee on message delivery. Message a is in transit while message b reaches process P_1. The predicate (Φ) is not satisfied at CUT1.

program location.

On the other hand, deterministic communication patterns would produce guaranteed results for distributed opaque predicates. An example of deterministic cyclic communication pattern is shown in Figure 7. This event-time diagram is a continuation of the one shown in Figure 4 and the vector clock ticks are continued along the timelines of processes P_1, P_2, and P_3. At $[14, 10, 9]$, process P_1 is ready to evaluate the distributed opaque predicate (Φ). Interestingly at this point, it can be guaranteed that there are no messages in transit and hence the predicate must hold true (Φ^T) at CUT2. For all other event intervals, (Φ) is guaranteed to be false (Φ^F).

3.5 Distributed opaque predicate embedding and guarded commands for maintaining local state consistency

Just as nondeterminism and asynchrony can be used as tools against the adversary, these could also cause problems to the obfuscator since uncontrolled concurrency will update states in an unpredictable way. If local states of processes are updated in an uncontrolled way, then distributed opaque predicates cannot be used effectively for control-flow obfuscation.

The problem associated with unpredictable local state update can be brought under control if the communication pattern generating code (specifically the *send/receive* primitives) can be guarded; i.e., a message contributing to a deterministic communication pattern is only sent from a process if it is guaranteed that the vector clock value of the process issuing this *send* is up-to-date. Alternatively, this means that the process should have completed all the message communication events (*send/receive*) before issuing another *send*. We show an abstract pseudo-code for controlled message passing and predicate evaluation for process P_1 in Figure 8. We have used blocking *receive* to ensure that the local state of process P_1 is consistent before it issues a *send* message (i.e., the process busy-waits on all outstanding messages it has not yet received). To make it more flexible, non-blocking *receive* with guarded *sends* could be used to maintain consistency of local states. This can be implemented by making sure that before each *send* primitive, the vector clock from last *receive* is up-to-date (by comparing it against an expected timestamp value). If the clock is not up-to-date, the process blocks the *send* call for outstanding *receives*.

Figure 8 shows pseudo-code snippets for nondeterministic (CUT1) and deterministic (CUT2) evaluation of the predicate Φ. At CUT1, Φ is unknown and hence dummy actions are inserted in branches corresponding to both 'true' and 'false' paths. However, at CUT2, the obfuscator knows that Φ holds true (because it knows when it participates in deterministic and nondeterministic message-passing) and hence inserts real actions in the path corresponding to the 'true' branch of the control statement. Pseudo process interaction codes for *guard* processes are similar to P_1's code and have been excluded from this contribution because of space limitations.

During obfuscation phase of P_1, the obfuscator may embed many distributed opaque predicates at different control-flow points in the program corresponding to, for example, the construction of watermarking code. Any arbitrary nesting of distributed opaque predicates can be used for obfuscating the control-flows. A different set of *guard* processes could also participate in different communication invariants involving other local state update rules.

4 Security analysis of obfuscation using distributed opaque predicates

In this section, we comment on the obfuscatory strength of the proposed technique by arguing that known forms of static analysis attacks and a restricted class of dynamic analysis attack are intractable from an adversary's perspective. For each class of attack, we also present our assumptions on technical limitations of the adversary.

4.1 Static analysis attacks

We argued in section 2 that static analysis of temporally unstable distributed opaque predicates will not reveal their outcome since the invariant communication pattern which influences their outcome is generated from the embedded *send/receive* primitives on-the-fly. We did not, however, comment on the difficult issues an attacker needs to address in order to statically *find* these distributed opaque predicates from the process codes.

In order to statically analyse the obfuscated code, an adversary must depend on static slicers to slice parts of the process code which could affect the value of distributed opaque predicates at obfuscated control-flow points. Slicing of distributed programs is a major challenge due to the timing related interdependencies among processes. Moreover, to find the slicing criterion of the slicer, the analyser must rely on alias analysis (Horowitz 1997, Hind *et al.* 1999) to determine the kind of structure the local state of processes points-to (this information he could get from the message parameters), and if the pointer corresponding to the variables used in the construction of distributed opaque predicates refer to the same dynamic data structure in the *guards* at some program location (where the distributed opaque predicates are used in P_1). To achieve this, static analysers must use inter-process escape alias analysis to determine the objects that can be referenced in processes separate from the ones in which they are allocated.

Though much research work on intra- as well as inter-procedural alias analysis and inter-procedural thread escape analysis have been done in the last few years (Rugina & Rinard 1999, Sălcianu & Rinard 2001, Whaley & Rinard 1999), we have been unsuccessful in finding a technique that can perform alias analysis by considering asynchronous message-passing of distributed processes as escape points. We believe the reason why this problem has not yet been

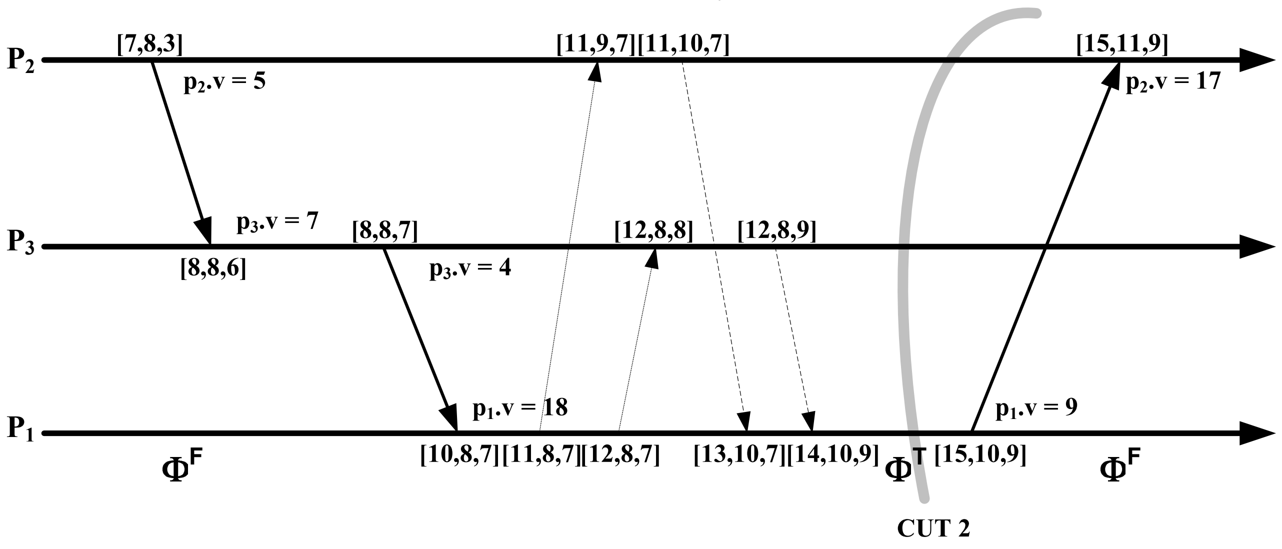


Figure 7: The invariant in the form of a predetermined deterministic communication pattern is shown in this figure. As before, thick arrows denote `SYSTEM` type messages, thin arrows denote `REQUEST` type messages, and dashed arrows denote `RESPONSE` type messages.

addressed by the program analysis community is because we do not have efficient, precise and scalable algorithms for performing simpler cases of alias analysis in sequential multi-threaded programs and asynchronous concurrent systems present problems that are much greater in magnitude.

4.2 Dynamic analysis attacks

Dynamic analysis attacks assume that the adversary has most (if not all) of the static analysis information available since he has to monitor the local state value changes of individual processes (an assumption we argued in the previous subsection as quite intractable). Moreover, in order to mount a dynamic analysis attack, the adversary needs to learn about the structure of the distributed opaque predicate so that he can identity which processes are contributing in building the global state (i.e. the *guard* processes along with process P_1). We need to make the restriction that an adversary cannot possess sufficient static analysis information in order to insert debugging probes at obfuscated control-flow points of P_1. If an adversary is able to do this, he can quite easily determine the outcome of distributed opaque predicates by just checking probe values during execution of process P_1. If we fail to make this restriction, the use of opaque predicates in any form of program obfuscation would be trivial. We, however, make the relaxation that the adversary can monitor the communication events using *sniffer* processes in order to monitor individual local states formed by process P_1 and the *guards*. This scenario is depicted in Figure 9.

We now show proceed to show that the problem of global state monitoring is hard even if the adversary manages to collect all necessary static analysis information. The problem of distributed opaque predicate evaluation, from the adversary's perspective, can be stated as evaluating predicate Φ as a function of the global state of a distributed system. It is problematic to detect unstable distributed opaque predicates since the condition encoded by the predicate may not persist long enough for it to be true when the predicate is evaluated by the adversary. The domain of such predicates is a Boolean valued function formed on the set of all possible cuts from all possible executions of the distributed system. Therefore, the predicate detection problem can also be defined as identifying a cut

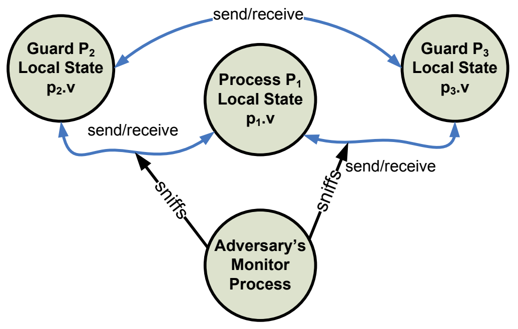

send/receive
Guard P2 Local State p2.v
Process P1 Local State p1.v
Guard P3 Local State p3.v
send/receive
send/receive
sniffs
sniffs
Adversary's Monitor Process

Figure 9: A typical dynamic analysis attack scenario by actively monitoring state changes to detect distributed opaque predicate Φ.

in which the predicate evaluates to true. The difficulty associated with detection is the fact that the number of states from any execution may be exponential in the number of processes.

Let $\{X_1, X_2, \ldots, X_n\}$ define a sequence of cuts, where for all i, $X_i < X_{i+1}$. A sequence of cuts is called an *observation* if and only if for all i, X_i and X_{i+1} differ by exactly one state. The adversary has to detect a consistent observation O of the distributed computation such that Φ holds in a global state of O. Now, this is a decision problem in the form of:

`Given:` an execution Y of n processes, an initial cut $X \leq Y$, and the predicate Φ.

`Determine:` if there exists a cut $W : X \leq W \leq Y$ such that $\Phi(W)$ is true.

Chase & Garg (1995) proved that this detection problem is NP-complete by showing that the detection of general global predicate is intractable even for simple distributed computation where the local states are restricted to take only `true` or `false` values and no messages are exchanged within the system.

In the subsections to follow, we model a dynamic analysis distributed monitoring attack by discussing in details the three types of available algorithms an adversary may choose to dynamically evaluate the outcome of Φ and the technical limitations these algorithms possess.

Process P_1:

```
initialize(VectorClock); //Initialize Vector Clock to [0,0,0]

… //Start nondeterministic predicate evaluation

//get SYSTEM message
while(!receive(VectorClockTimeStamp,SYSTEM,bufferVal)){
    probe(ReceivePort); //Check for message by polling
}
increment(VectorClock);
setMax(VectorClock, VectorClockTimeStamp); // Vector Clock value [1,1,0]
shift_right($p_1$.v); //point to the right node

//probe for local states from guard processes
increment(VectorClock); // Vector Clock value [2,1,0]
send($P_2$,REQUEST); //probe for $p_2$.v value
increment(VectorClock); // Vector Clock value [3,1,0]
send($P_3$,REQUEST); //probe for $p_3$.v value

//get RESPONSE messages
while(!receive(VectorClockTimeStamp,RESPONSE,bufferVal)){
    probe(ReceivePort); //Check for message by polling
}
$p_2$.v = bufferVal;
increment(VectorClock);
setMax(VectorClock, VectorClockTimeStamp); // Vector Clock value [4,4,1]
while(!receive(VectorClockTimeStamp,RESPONSE,bufferVal)){
    probe(ReceivePort); //Check for message by polling
}
$p_3$.v = bufferVal;
increment(VectorClock);
setMax(VectorClock, VectorClockTimeStamp); // Vector Clock value [5,4,3]

// evaluate distributed opaque predicate
if ($p_1$.v+$p_2$.v+$p_3$.v==27) {// CUT1: Predicate Value Unknown
    // Dummy watermark building code
}
else {
    // Dummy watermark building code
}

… //Start deterministic predicate evaluation

//get SYSTEM message
while(!receive(VectorClockTimeStamp,SYSTEM,bufferVal)){
    probe(ReceivePort); //Check for message by polling
}
increment(VectorClock);
setMax(VectorClock, VectorClockTimeStamp); // Vector Clock value [10,7,8]
shift_right($p_1$.v); //point to the right node

//probe for local states from guard processes
increment(VectorClock); // Vector Clock value [11,8,7]
send($P_2$,REQUEST); //probe for $p_2$.v value
increment(VectorClock); // Vector Clock value [12,8,7]
send($P_3$,REQUEST); //probe for $p_3$.v value

//get RESPONSE messages
while(!receive(VectorClockTimeStamp,RESPONSE,bufferVal)){
    probe(ReceivePort); //Check for message by polling
}
$p_2$.v = bufferVal;
increment(VectorClock);
setMax(VectorClock, VectorClockTimeStamp); // Vector Clock value [13,10,7]
while(!receive(VectorClockTimeStamp,RESPONSE,bufferVal)){
    probe(ReceivePort); //Check for message by polling
}
$p_3$.v = bufferVal;
increment(VectorClock);
setMax(VectorClock, VectorClockTimeStamp); // Vector Clock value [14,10,9]

// evaluate distributed opaque predicate
if ($p_1$.v+$p_2$.v+$p_3$.v==27) {   // CUT2: Predicate Value True
    // Real watermark building code
}
else {
    // Dummy watermark building code
}
```

Figure 8: Pseudo-code showing the obfuscation of control-flow in P_1 using distributed opaque predicate Φ.

4.2.1 Active monitoring by taking snapshot

The first option the adversary has is to solve the global predicate evaluation problem through *active* monitoring. In this strategy, the adversary uses a monitor process which sniffs the communication between process P_1 and the *guards* at some predetermined periodic intervals and then combines all the local states obtained to build the global state. This strategy is called *'snapshot'* approach and Chandy & Lamport (1985) describe an algorithm to construct consistent global states using snapshots of individual local states. Since communication within distributed systems incurs latency, the consistent global states thus constructed can only reflect some past state of the system. By the time the snapshots are obtained, conclusions drawn about the system by evaluating the distributed opaque predicate may have no bearing to the present. Therefore, the snapshot algorithm is suitable for monitoring predicates that do not change value throughout the entire program run and since our method uses temporally unstable predicates, the adversary will not be able to deduce a correct reasoning about the predicate's behaviour using this algorithm - the predicate may have held even if it is not detected.

4.2.2 Passive monitoring by constructing state lattice

Through the second approach, due to Cooper & Marzullo (1991), the adversary can collect all local state values from individual processes and check for consistent observation O using *passive* monitoring. In order to implement this algorithm, the adversary's monitoring process must sniff the *guard* processes and P_1 for portions of their local states that are referenced in Φ. The monitor maintains sequences of these local states, one sequence per process, and uses them to construct the global state. This procedure is based on incrementally constructing the lattice of consistent global states associated with the distributed computation. The state lattice formed is linear in the number of global states, and the number of global states formed is $O(e^n)$ where e is the maximum number of events monitored and n is the number of processes in the system. For every global state in the lattice, there exists at least one run that passes through it. Hence, if any global state in the lattice satisfies Φ, the distributed opaque predicate is detected.

The problem with this type of monitoring is that the adversary may end up incorrectly including spurious local state changes in case he erroneously considers processes that are interacting with *guards* and P_1 during construction of the lattice or if he includes spurious message communication events (by failing to distinguish between message classes that only update the vector clock values and not the local state of processes). Hence, if the number of *guards* in the system is large and a considerable amount of message exchange takes place, the adversary will face the problem of state explosion while trying find a consistent cut by 'walking-through' the lattice thus formed. Moreover, if the adversary fails to monitor some of the process interactions, the amount of concurrency in the form of local state changes will increase and this will, in turn, increase the states of the lattice. Increase in the number of *guards* in the system will increase the dimension of the lattice proportionately. Furthermore, the adversary has to repeat this passive monitoring process to detect the outcome of each distributed opaque predicate used to obfuscate control-flow points in process P_1. The complexity will further increase if such predicates are nested and *guards* are spawned dynamically during P_1's execution.

4.2.3 Active monitoring by exploiting predicate structure

In the final approach, the adversary can use Garg & Waldecker's (1994) method for detecting unstable global predicates. Their method exploits the structure of predicate by decomposing the predicate into a conjunction of local predicates and independently detecting the outcomes of these local predicates. It also requires the use of explicit token passing messages between the monitor process and the processes which contribute states in the construction of distributed opaque predicates. This approach works well for predicates that are conjunctive in nature. However, for relational distributed opaque predicates, their method yields no feasible solution because relational predicates cannot be broken down into conjunction of predicates formed on local states. Moreover, it requires processes participating in maintaining the distributed opaque predicate to cooperate with adversary's monitor process by maintaining snapshots (evaluating their component of the predicate) and passing the result and dependence information to the adversary's monitoring process. This requirement quite unrealistic under reasonable practical assumptions.

We conclude by observing that out of these three available approaches, the adversary has to resort to using only the second approach because the other two available approaches are only suitable for detecting either predicates that do not change their value during the entire program run or predicates that can be broken down into a conjunction of local predicates. Moreover, the second approach will be intractable if a large number of *guard* processes are used or are spawned dynamically during execution of the obfuscated process P_1. Also, in the absence of precise static analysis methodologies, spurious events and state changes would be erroneously taken into consideration by the adversary and this would make the detection process incorrect and intractable. Under pragmatic assumptions, we believe practical distributed systems will employ a large number of processes as *guards* and hence processes obfuscated with distributed opaque predicates will be resilient to passive monitoring dynamic analysis attacks.

5 Conclusion

In this contribution, first of its kind, we have addressed the problem of code obfuscation in software executing in distributed computing environments. Specifically, we have addressed control-flow obfuscation and have extended the original concept of opaque predicates proposed by Collberg *et al.* (1998) to the domain of distributed computing. We have demonstrated that hard combinatorial problems can be tuned with open problems related to distributed systems state monitoring to manufacture a new class of resilient opaque predicates; which we defined in this contribution as distributed opaque predicates. We have also demonstrated through a detailed security analysis that our class of distributed opaque predicates is resilient to known static analysis attacks and passive monitoring dynamic analysis attack from adversaries.

The following salient points could be noted regarding this new class of opaque predicates:

- `Stealth:` The relational structure of distributed opaque predicates will make these unobvious to an attacker since predicates of this nature appear as conditional expressions in most programs. Moreover, *guard* processes, along with the process to be obfuscated, maintain the distributed opaque predicate invariant through an embedded communication pattern. This pattern is generated by *send/receive* calls embedded within process code. Processes in loosely-coupled distributed systems, as such, communicate using message-passing and hence the presence of additional *guards* and their interactions with the host process will, we believe, be unsuspecting from the perspective of an adversary.

- `Performance:` Distributed systems are inherently loosely-coupled in nature and do not enforce hard timing requirements on task completion. Hence, the overall slowdown in system effectuated by additional *guard* processes and message exchanges might be acceptable to developers having stringent security requirements.

As part of our future work, we will concentrate on automatic embedding of distributed opaque predicates at selected control-flow locations in distributed computing processes through program annotation. We would also come up with a model of making the obfuscated system, consisting of the obfuscated process and cooperating *guards*, more fault-tolerant such that the system can function in case one or more *guards* are accidentally lost or purposefully killed by an adversary. Our present model is rigid in the sense that the loss of *guard* processes will make the obfuscated program go into an incorrect state, thus adding some form of tamper-proofing. But, this notion is weak since the *guards* and messages may be lost in the system accidentally. We will also investigate into new classes of distributed opaque predicates and instances of hard combinatorial problems for generating them in the future.

References

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., & Yang, K. (2001), On the (Im)possibility of Obfuscating Programs, *In* the proceedings of CRYPTO-2001. LNCS Volume 2139, Springer-Verlag. Santa Barbara, CA, USA

Chow, S., Gu, Y., Johnson, H. & Zakharov, V.A. (2001), An Approach to the Obfuscation of Control-Flow of Sequential Computer Programs. *In* the proceedings of 4th International Conference on Information Security, LNCS Volume 2200. Springer-Verlag. Malaga, Spain.

Garey, M. R. & Johnson, D. S. (1979), A guide to the theory of NP-completeness. W.H. Freeman and Company.

Thomborson, C., Nagra, J., Somaraju, R. & He, C. (2004), Tamper-proofing software watermarks. *In* the proceedings of 2nd workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalization. Volume 32. Dunedin, New Zealand. ACM Digital Library.

SETI@home (2005), http://setiathome.ssl.berkeley.edu/ (accessed July 15 2005).

Palsberg, J., Krishnaswamy, S., Kwon, M., Ma, D., Shao, Q. & Zhang, Y. (2000), Experience with software watermarking. *In* the proceedings of 16th IEEE Annual Computer Security Applications Conference (ACSAC'00). IEEE Press. New Orleans, LA, USA.

Nagra, J. & Thomborson, C. (2004), Threading Software Watermarks. *In* the proceedings of 6th International Workshop on Information Hiding, LNCS Volume 3200, Springer-Verlag. Toronto, ON, Canada.

Hohl, F. (1998), Time limited blackbox security: Protecting mobile agents from malicious hosts. *In* the proceedings of 2nd International Workshop on Mobile Agents, LNCS Volume 1419, Springer-Verlag. Stuttgart, Germany.

Collberg, C., Thomborson, C. & Low, D. (1998), Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs. *In* the proceedings of 1998 ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'98). San Diego, CA, USA.

Lamport, L. (1978), Time, clocks and the ordering of events in a distributed system. *In* Communications of the ACM, 21(7):558-565.

Chase, C. & Garg, V.K. (1995), Detection of global predicates: Techniques and their limitations. *In* the Journal of Distributed Computing, Volume 11, Issue 4, pages 191 - 201. Springer-Verlag.

Mattern, F. (1989), Virtual time and global states of distributed systems. *In* the proceedings of Workshop on Parallel and Distributed Algorithms, Elsevier Science Publication, pages 215-226.

Horowitz, S. (1997), Precise Flow-insensitive may-alias in NP-hard. *In* ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 19 No. 1.

Hind, M., Burke, M., Carini, P. & Choi, J.D. (1999), *In* ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 21 No. 4.

Rugina, R. & Rinard, M. (1999), Pointer analysis for multithreaded programs. *In* the proceedings of 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99). Atlanta, GA, USA.

Sălcianu, A. & Rinard, M. (2001), Pointer and escape analysis for multithreaded programs. *In* the proceedings of 2001 ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP '01), Snowbird, UT, USA.

Whaley, J. & Rinard, M. (1999), Compositional pointer and escape analysis for Java programs. *In* the proceedings of 1999 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA '99), Denver, CO, USA.

Chandy, K.M. & Lamport, L. (1985), Distributed Snapshots: Determining global states of distributed systems. *In* ACM Transactions on Computer Systems, pages 63-75.

Cooper, R. & Marzullo, K. (1991), Consistent detection of global predicates. *In* the proceedings of 1991 ACM/ONR Workshop on Parallel and Distributed Debugging (PADD '91). Santa Cruz, CA, USA.

Garg, V. K. & Waldecker, B. (1994), Detection of weak unstable predicates in distributed programs. *In* IEEE Transactions on Parallel and Distributed Systems, pages 299-307, Volume 5, No. 3.

Pruning Subscriptions in Distributed Publish/Subscribe Systems

Sven Bittner **Annika Hinze**

Department of Computer Science
University of Waikato,
Private Bag 3105, Hamilton, New Zealand,
Email: {s.bittner, a.hinze}@cs.waikato.ac.nz

Abstract

Publish/subscribe systems utilize filter algorithms to determine all subscriptions matching incoming event messages. To distribute such services, subscriptions are forwarded to several filter components. This approach allows for an application of routing algorithms that selectively forward event messages to only a subset of filter components. Beneficial effects of this scheme include decreasing network and computational load in single filter components.

So far, we can find routing optimizations that exploit coverings among subscriptions or utilize subscription merging strategies. Generally, such optimizations aim at reducing the amount of subscriptions forwarded to filter components, which decreases their computational load. This might in turn result in an increasing number of event messages routed through the network.

However, current optimization strategies only work on restrictive conjunctive subscriptions and cannot be extended to efficiently support arbitrary subscriptions. Furthermore, it is not possible to apply covering and perfect merging strategies in all application scenarios due to the strong dependency of these approaches on actually registered subscriptions.

In this paper, we present a novel optimization approach, subscription generalization, to decrease the filtering overhead in publish/subscribe systems. Our approach is based on selectivities of subscriptions and can be utilized for all kinds of subscriptions including arbitrary Boolean and conjunctive subscriptions. We propose a simple subscription generalization algorithm and show an evaluation of the results of a first series of experiments proving the usefulness of our approach.

Keywords: Distributed publish/subscribe, event filtering, subscription tree pruning

1 Introduction

Publish/subscribe systems use a push-based approach to access information that is published in the form of event messages. This means these systems continuously filter and actively deliver information to interested parties, which register their interests by the help of subscriptions. We can effectively apply publish/subscribe systems for several applications, e.g., facility management (Hinze 2003), meteorology (Mathieson, Dance, Padgham, Gorman & Winikoff 2004), healthcare (Jung & Hinze 2005) and electronic commerce (Cilia & Buchmann 2002).

To achieve large-scale publish/subscribe solutions, these systems have to be realized as distributed services (Mühl 2002). They consist of several broker components dividing the filter load and thereby collaboratively performing the overall filtering task.

We have illustrated such a distributed publish/subscribe system in Figure 1: In the simplest case broker components B_x are connected by an acyclic (overlay) network structure. Clients, i.e., publishers P_x, subscribers S_x, and clients acting as both parties, connect to an arbitrary broker. This broker is called local broker and hides the distributed nature of the system, e.g., B_2 is local broker for P_1 in Figure 1. Subscriptions s_x are registered at the respective local brokers; event messages e_x are published to them. Matching event messages are delivered to subscribers by their local broker by the help of notifications n_x.

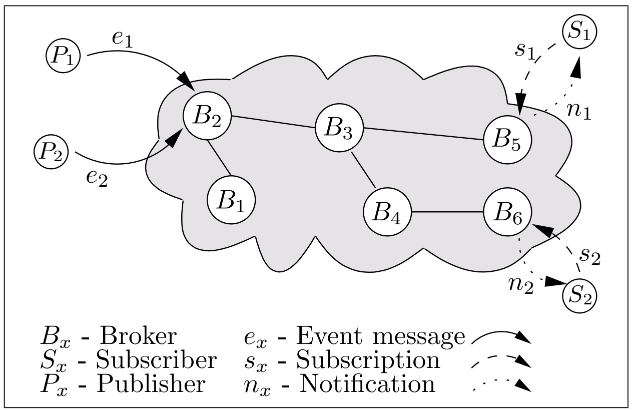


Figure 1: Distributed publish/subscribe system

In most application areas for publish/subscribe systems, the frequency of event messages is much higher than the frequency of registering and deregistering subscriptions. Thus, a profile forwarding scheme (Bittner & Hinze 2004) should be utilized to reduce the amount of event messages forwarded to brokers: Subscriptions are forwarded from local brokers to neighbor brokers. Then, brokers only deliver event messages to neighbor brokers that fulfil their subscriptions (which could again have been registered with another broker component). This forwarding of event messages and subscriptions is also referred to as event and subscription routing, respectively.

To minimize the amount of forwarded subscriptions, current approaches utilize coverings among subscriptions or merge several subscriptions. The main drawbacks of these methods are that their efficacy strongly depends on registered subscriptions and that they are applicable to conjunctive subscriptions only.

Especially in advanced application areas (e.g., e-commerce) the constraint of conjunctive subscriptions is too restrictive and does not allow for the definition of subscriptions required to specify user interests.

In Figure 2, we present an example subscription s_1 from an e-commerce setting, in particular from online book auctions (letters in the figure describe the name of nodes). We will use s_1 as a running example throughout this paper: A subscriber is interested in books whose title contains the phrase "Harry Potter". According to the condition of the copy of the book (new, used), she wants to pay a different price (at most NZ$15.0 or NZ$10.0, respectively). To avoid unnecessary notifications, the subscriber will be notified not earlier than one day before the auction ends.

We can represent subscriptions as subscription trees as shown in Figure 2. Inner nodes represent Boolean operators; leaf nodes specify predicates, i.e., attribute-operator-value triples (Bittner & Hinze 2005*b*).

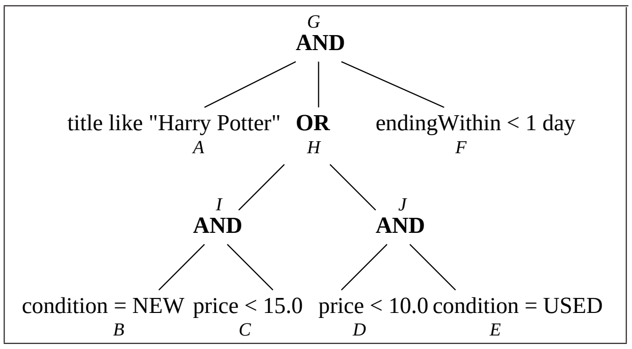


Figure 2: Example of a Boolean subscription s_1

Next to this requirement for more expressive than pure conjunctive subscriptions, it has been shown that publish/subscribe systems supporting more expressive subscription languages retain the efficiency properties of systems only offering conjunctive languages. In fact, the utilization of more expressive languages increases the scalability properties of brokers performing event filtering (Bittner & Hinze 2005*a*).

These beneficial characteristics of systems supporting arbitrary Boolean subscriptions (including conjunctive subscriptions) lead to the necessity of developing routing optimizations that are applicable to such services. Since we cannot use current covering and merging approaches in conjunction with expressive subscription languages, in this paper we design a novel routing optimization approach, subscription generalization, working on arbitrary subscriptions.

Subscription generalization is based on selectivities of subscriptions and aims at reducing subscription complexity to relieve resources in filtering broker components. Thus, it decreases their computational load as well as their memory requirements. This, in turn, results in increasing efficiency and scalability properties in broker components themselves.

However, this beneficial effect is counteracted by increased network load when applying subscription generalization. This behavior originates in the decrease of selectivities of subscriptions due to generalizations. Hence, more event messages are forwarded to neighbor brokers.

The rest of this paper is structured as follows: In Section 2 we present related work including routing optimization and selectivity estimation algorithms. The overall idea behind subscription generalization is elaborated in Section 3. There we also outline two specific generalization approaches, subscription pruning (Section 3.2) and predicate replacement (Section 3.3). Section 4 presents our proposal to estimate selectivities of subscriptions. Then, we continue with an automatic generalization algorithm in Section 5. Section 6 presents a series of experiments we have undertaken to evaluate our approach as well as the evaluation of their results. Finally, we conclude and present future work in Section 7.

2 Current Optimization Approaches

We can classify most current routing optimization approaches into covering-based and merging-based solutions. These approaches are presented in the following subsections. We discuss current proposals on estimating selectivity in Section 2.4.

2.1 Subscription Covering

The definition of coverings among subscriptions, e.g., s_2 and s_3, is based on analyzing the sets of event messages fulfilling them. These sets of fulfilling event messages are referred to as $E(s_2)$ and $E(s_3)$, respectively. If it holds $E(s_2) \supseteq E(s_3)$ then subscription s_2 covers s_3 (Mühl & Fiege 2001).

Coverings have been investigated in the publish/subscribe systems SIENA (Carzaniga, Rosenblum & Wolf 2001) and REBECA (Mühl & Fiege 2001). Both systems only support conjunctive subscriptions. Furthermore, (Mühl & Fiege 2001) restricts subscriptions to contain at most one predicate per attribute; (Carzaniga et al. 2001) does not present algorithms to compute coverings at all.

Also XML-based publish/subscribe systems, e.g., XROUTE (Chand & Felber 2003), utilize subscription covering. However, these systems restrict their subscription languages to conjunctive forms, too.

For general database systems, research also addresses a problem similar to covering: query rewriting (Halevy 2000). Since, in contrast to publish/subscribe systems, database systems can effectively utilize queries in canonical forms (Bittner & Hinze 2005*b*), research for database systems only targets these cases. Finding the required rewriting for general conjunctive database queries is NP-complete (Halevy 2000).

Generally, the utilization of coverings among subscriptions heavily depends on these registered subscriptions, i.e., we can only optimize by the help of coverings if subscriptions cover each other. The amount of optimization achievable by coverings is determined by the covering properties of subscriptions. So far, there is no research evaluating these properties in real-world applications.

2.2 Subscription Merging

Another optimization technique is subscription merging. There a set of subscriptions $\mathcal{S}_4$ is merged into a smaller set of subscriptions $\mathcal{S}_5$ with $|\mathcal{S}_4| > |\mathcal{S}_5|$ and $\bigcup_{s_4 \in \mathcal{S}_4} E(s_4) \subseteq \bigcup_{s_5 \in \mathcal{S}_5} E(s_5)$. In case of set equality we refer to this as perfect merging; for a proper subset relation it is denoted as imperfect merging.

Perfect merging does not increase the network traffic for event routing compared to originally registered subscriptions (Wang, Qiu, Verbowski, Achlioptas, Das & Larson 2004). However, we cannot always find such a merging; its applicability depends on the kind of registered subscriptions. REBECA (Mühl 2001) supports perfect merging for restricted conjunctive queries as described above. The work in (Crespo, Buyukkokten & Garcia-Molina 2003) analyzes several merging strategies for geographic queries, i.e., simple two-predicate conjunctive queries.

As pointed out in (Wang et al. 2004), perfect merging may not optimize the overall system throughput. Therefore, this work utilizes imperfect merging. It clusters subscriptions according to similarity to achieve a compact summary (merging) of them and evaluates several clustering techniques. Again, only conjunctive subscriptions are supported.

A general objection to subscription merging is its benefit when applying to subscriptions converted into conjunctive forms (arbitrary Boolean subscriptions might be converted into disjunctive normal forms of exponential size). Considering imperfect merging, we do not face the sole dependency on registered subscriptions as in the covering approach. However, the most possibilities to perform merging might result out of converted subscriptions, i.e., subscriptions previously converted and split into conjunctive elements are merged into a more general (conjunctive) subscription. This questions the usefulness of canonical conversions in respect to routing optimizations in addition to their drawbacks of decreasing scalability in broker components (Bittner & Hinze 2005*a*).

2.3 Other Optimization Approaches

Several approaches, e.g., the proposal in (Guimarães & Rodrigues 2003), target the routing in publish/subscribe systems as a multicast mapping problem: Similar subscriptions are clustered in the same multicast group. Event messages are only sent to those groups that may contain matching subscriptions. However, such approaches are also restricted to conjunctive subscription languages.

The routing scheme in (Carzaniga, Rutherford & Wolf 2004) mentions subscription simplification, which is a similar approach to subscription merging (perfect as well as imperfect). However, the work presents no computation algorithms. It supports subscriptions in disjunctive normal form and thus requires the same conversions as conjunctive approaches if it is utilized in applications involving more expressive than conjunctive subscriptions.

2.4 Selectivity Estimation

Estimating the selectivity of queries has been researched in the context of database systems, e.g., (Poosala & Ioannidis 1997, Chen, Koudas, Korn & Muthukrishnan 2000). However, such approaches require either conjunctive queries or conversions of queries into disjunctive or conjunctive normal forms. Apart from the time complexity required for these selectivity estimations and the memory consumption of involved data structures, canonical conversions lead to exponential space complexity that is not applicable in the context of publish/subscribe systems (Bittner & Hinze 2005*a*). These solutions are applicable to database systems that do not have to deal with a large number of continuous queries (another term used for subscriptions) and thus can convert queries into canonical forms.

3 Subscription Generalization

The overall idea of generalizing subscriptions is to decrease the computational effort for event filtering required in broker components. In the next subsection, we outline our general approach and its effect on event filtering. Afterwards, we present two simple subscription generalization methods, pruning subscription trees and replacing predicates by more general ones, i.e., covering predicates. We describe these methods in Section 3.2 and Section 3.3, respectively.

We discuss subscription generalization if utilizing one-dimensional index structure-based filter algorithms, e.g., (Bittner & Hinze 2005*b*, Fabret, Jacobsen, Llirbat, Pereira, Ross & Shasha 2001, Hanson, Chaabouni, Kim & Wang 1990, Pereira, Fabret, Llirbat & Shasha 2000, Yan & García-Molina 1994). Such approaches utilize one-dimensional indexes to index predicates and propose different structures to index subscriptions. Event filtering works in two steps: In predicate matching, all predicates matching incoming event messages are determined. Then, subscription matching computes all matching subscriptions (based on matching predicates). Since we focus on Boolean subscriptions, we particularly consider the algorithm in (Bittner & Hinze 2005*b*) that uses tree structures as subscription indexes.

3.1 Purpose of Subscription Generalization

As stated before, most application areas for publish/subscribe systems can effectively apply a subscription forwarding scheme. That is, subscriptions are forwarded to neighbor brokers of the local broker they have been registered with. This allows broker components to determine the set of neighbor brokers each incoming event message e_x should be forwarded to. In case of acyclic network structures this set includes all neighbors with registered subscriptions that are fulfilled by e_x except the neighbor forwarding e_x.

Subscription generalization aims to decreasing the complexity of forwarded subscriptions. Whenever such subscriptions require too many memory resources, broker components start to heuristically generalize registered subscriptions in order to minimize their size and thus the size of indexing structures. We do not generalize subscriptions registered by clients to avoid false notifications; only subscriptions forwarded by neighbor brokers are suitable for generalizations.

The generalization of subscriptions decreases both the memory usage and the computational effort required in brokers to determine matching subscriptions: The complexity of subscriptions is decreased, i.e., subscriptions consist of less predicates and operators, which directly decreases the memory required for subscription index structures. Furthermore, when indexes become smaller, we can determinate matching subscriptions more efficiently: In one-dimensional indexing approaches the predicate matching step works on smaller numbers of predicates; subscription matching filters on less complex subscriptions.

However, this advantageous behavior increases network load due to the forwarding of more event messages to neighbor brokers (subscriptions are more general now). This property is consistent with imperfect merging, which, similar to subscription generalization, does not depend on registered subscriptions like the proposals of perfect merging and covering. Thus, subscription generalization does always lead to decreasing memory requirements and increasing efficiency in broker components. We analyze the correlation between memory usage and network load in our experiments in Section 6.

A subscription generalization method for publish/subscribe systems should be easy to compute to allow for an efficient registering and deregistering of large numbers of subscriptions. Furthermore, it should require little additional memory to retain scalability properties of broker components.

In the next subsections, we describe two particular subscription generalization methods, subscription tree pruning and predicate replacement.

3.2 Generalization by Pruning

Pruning subscription trees reduces the number of predicates a broker has to filter on in the predicate matching step. Furthermore, it reduces the complexity of subscriptions that need to be evaluated in subscription matching. This characteristic is obtained by pruning certain branches of subscription trees. Notice that we do not prune subscriptions in local brokers (cf. Section 3.1).

An arbitrary pruning of subscription trees does not necessarily lead to more general subscriptions. A subscription s_x is more general than subscription s_y if $E(s_x) \supseteq E(s_y)$. This definition conforms with the definition of covering for conjunctive subscriptions known from literature, e.g., (Mühl & Fiege 2001).

Excluding the case of removing the root node in a subscription tree, there is only one kind of pruning leading to a more general subscription:

Remove a child of a conjunctive node in a subscription tree.

An example is given in Figure 3. There we have illustrated the same subscription s_1 as in Figure 2 when removing node D that is a child of conjunctive node J. This pruning operation leads to s_6, which, in contrast to original subscription s_1, describes interest in all used books conforming to the other criteria (`Title` and `Ending Within`).

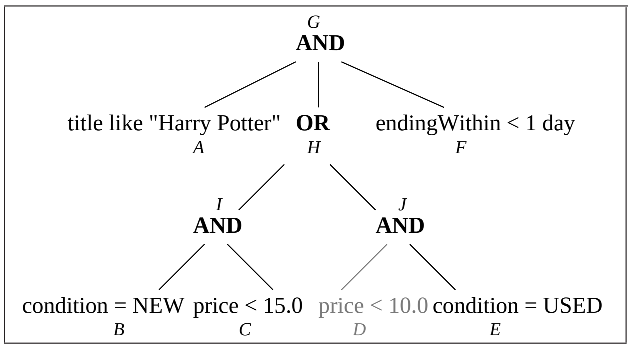


Figure 3: Valid pruning of s_1 leading to s_6

Removing a child of a disjunction is not a valid pruning as shown in Figure 4(a). There we show subscription s_1 (cf. Figure 2) when removing node J, the child of a disjunction. After restoring the general subscription tree properties (subsuming consecutive operators and eliminating unary operators (Bittner & Hinze 2005*b*)), the new subscription s_7 consists of only one conjunction as shown in Figure 4(b). However, it holds $E(s_7) \not\supseteq E(s_1)$, e.g., any event message describing used books does not fulfil s_7 but s_1. Thus, this removal of node J does not lead to a more general subscription and is invalid.

However, our valid pruning option includes removing a complete disjunction in subscription trees: Disjunctions are always (except if they are a root node) the child of a conjunction due to our subsuming of consecutive operators. Hence, this option of removing a disjunction is included in our single pruning rule.

Furthermore, we can apply subscription tree pruning to pure conjunctive subscriptions. We are able to perform pruning in all cases, which makes our approach beneficial compared to current covering proposal that strongly depend on registered (conjunctive) subscriptions.

3.3 Generalization by Replacement

Our second proposal to generalize subscriptions is to replace predicates p_x by more general predicates p_y, i.e., p_x is covered by predicate p_y. The determination of coverings among predicates is relatively easy to calculate and depends on the permitted operators. Some examples are given in (Mühl 2001). However, the overall approach of subscription covering (Mühl 2001) is not applicable to arbitrary Boolean subscriptions.

The replacement of predicates results in decreasing numbers of predicates to filter on in the predicate

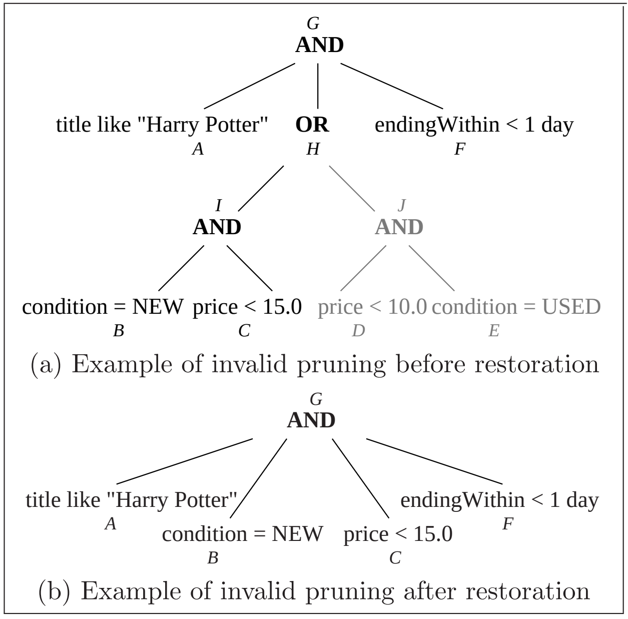


Figure 4: Invalid pruning of s_1 leading to s_7

matching step. However, it does not relieve subscription matching since subscriptions retain their complexity.

An example for the exploitation of coverings (based on s_1 that is given in Figure 2) is the replacement of predicate D by *price* $<$ 15.0 leading to s_8.

Replacements might also allow us to further modify resulting subscriptions. However, such modifications involve semantic knowledge (e.g., for subscription s_8 books are always new or used), which we neglect in our approach out of computational complexity and space efficiency aspects that are crucial in publish/subscribe systems (Bittner & Hinze 2005*a*).

We also do not focus on standard transformation rules, e.g., the integration of predicate *price* $<$ 15.0 of s_8 into the conjunctive root node G. We could integrate such transformations into our approach but abstract from them in this paper to evaluate the effect of generalizations themselves.

3.4 Interconnection between Pruning and Replacement

A more generalized view on the presented pruning and replacement methods reveals their interconnection: Pruning could be seen as a replacement of predicates or whole subtrees by the most general predicate p_*, which is fulfilled by each event message.

In a conjunctive node we can omit a child p_*, which is fulfilled by all event messages, without changing the set of events fulfilling this conjunctive expression. This behavior is described by our pruning rule.

If a child of a disjunctive node is the most general predicate p_*, the whole disjunction is fulfilled by all event messages. Thus, we can replace this disjunctive node by p_*. In turn, we are able to remove p_* since it is child of a conjunctive node (if it is not a root itself). This describes the removal of a disjunctive node.

However, in this paper we distinguish between pruning and replacement because the computational effort required for their calculation is rather different (Section 5).

After this introduction to subscription generalization and two particular generalization methods, we introduce our method of estimating the selectivity of subscriptions in the next section. This estimation is an integral part of our automatic generalization algorithm, which is presented afterwards in Section 5.

4 Estimated Selectivity

Our later proposal for automatic subscription generalization (Sect. 5) is merely based on the selectivity of predicates. Predicate selectivity allows us to successively estimate the selectivity of a subscription.

4.1 Calculation of Estimated Selectivity

We can calculate the selectivity of predicates based on historical information: For each predicate we maintain a counter that is increased whenever this particular predicate is fulfilled by an incoming event message[1]. Dividing this counter by the total number of filtered event messages leads to the selectivity of predicates. For newly registered subscriptions involving unused predicates we can estimate their selectivity, e.g., as presented in (Guimarães & Rodrigues 2003). Thus, for predicates p_x we can compute their selectivity $sel(p_x)$ both space and time efficiently.

The calculation of the selectivity $sel(s_x)$ of general Boolean subscriptions s_x would involve a huge computational effort requiring the analysis of all incoming event messages in respect to all registered subscriptions. Hence, we should focus on a simple method to estimate the selectivity $sel^{\approx}(s_x)$ of subscriptions s_x.

Our selectivity estimation $sel^{\approx}(s_x)$ for a subscription s_x consists of three easily computable values describing the minimal possible $sel^{min}(s_x)$, the average $sel^{avg}(s_x)$ (assuming a uniform distribution of all possible event messages and independent attribute values as well as predicates) and the maximal possible $sel^{max}(s_x)$ selectivity:

$$sel^{\approx}(s_x) = (sel^{min}(s_x), sel^{avg}(s_x), sel^{max}(s_x))$$

For all predicates p_x we can calculate their selectivity as described at the beginning of this subsection (count fulfilling event messages) and it holds

$$sel^{min}(p_x) = sel^{avg}(p_x) = sel^{max}(p_x) = sel(p_x)$$

We can recursively compute our selectivity estimation based on Boolean operators. For a binary conjunction the calculation works as follows

$$\begin{aligned} sel^{min}(a \wedge b) &= \min(0, sel^{min}(a) + sel^{min}(b) - 1) \\ sel^{avg}(a \wedge b) &= sel^{avg}(a) * sel^{avg}(b) \\ sel^{max}(a \wedge b) &= \min(sel^{max}(a), sel^{max}(b)) \end{aligned}$$

A binary disjunction leads to

$$\begin{aligned} sel^{min}(a \vee b) &= \max(sel^{min}(a), sel^{min}(b)) \\ sel^{avg}(a \vee b) &= sel^{avg}(a) + sel^{avg}(b) - \\ &\quad sel^{avg}(a) * sel^{avg}(b) \\ sel^{max}(a \vee b) &= \min(1, sel^{max}(a) + sel^{max}(b)) \end{aligned}$$

Our filtering approach for Boolean subscriptions (Bittner & Hinze 2005*a*, Bittner & Hinze 2005*b*) subsumes consecutive binary operators in subscription trees to n-ary ones. We can generalize the former calculations for the binary case to work with n-ary operators.

Calculating the described estimations for all subtrees of a subscription tree of s_x finally leads to the required selectivity estimate $sel^{\approx}(s_x)$. We have illustrated the estimate for our example subscription s_1 in Figure 5. Selectivities are rounded in the figure; we have given the selectivity of predicates only once (they are derived from our experiments whose setup is described in Section 6.1).

[1]For shared predicates this is done only once.

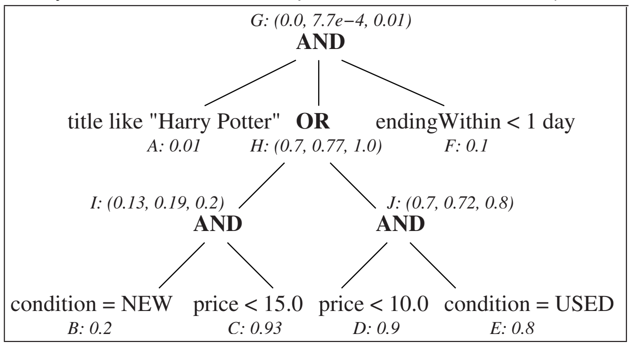


Figure 5: Example of estimating selectivity for s_1

4.2 Meaning of Estimated Selectivity

Our selectivity measure $sel^{\approx}(s_x)$ is an estimate of the real selectivity $sel(s_x)$ of s_x. It holds for all distributions of values in event messages and for all dependencies among attributes.

The value $sel^{max}(s_x)$ always describes the case that $E(s_x)$ is maximal: In case of a binary conjunction, the smaller set of event messages fulfilling a subexpression is a subset of the larger set of fulfilling event messages. For a binary disjunction, both sets are disjoint.

On the contrary, $sel^{min}(s_x)$ describes the case of a minimal $E(s_x)$: For a binary conjunction, the sets of fulfilling event messages of subexpressions exclude each other to the maximal possible extend. In case of a binary disjunction, the smaller set is included in the larger one.

Finally, $sel^{avg}(s_x)$ assumes that all possible event messages are equiprobable and subexpressions of subscriptions do not depend on each other: For a binary conjunction, the selectivity of one subexpression holds for event messages fulfilling the other subexpression. The same is true for the disjunctive case.

The real selectivity $sel(s_x)$ of s_x is always located between the two estimated extremes $sel^{min}(s_x)$ and $sel^{max}(s_x)$. The average case $sel^{avg}(s_x)$ describes which extreme is more likely if assuming independent predicates. For the subtree rooted in the disjunctive node H in Figure 5, it holds

$$sel^{\approx}(H) \quad = \quad (0.7, 0.77, 1.0)$$

Thus, the described average case is nearer to the estimated minimal than to the maximal selectivity value.

Our selectivity measure allows us to develop an automatic generalization algorithm targeting at the generalization of subscriptions according to the estimated effect on selectivity. This directly influences the increase in network traffic for event routing. We present this algorithm in the next section.

5 Automatic Generalization Algorithm

Our optimization algorithm targets an automatic generalization of subscriptions. Our goal is to decrease the computational costs of event filtering without increasing the network traffic to a large extend.

To achieve this goal, we generalize subscriptions s_x to s_y, which, in turn, might degrade their real selectivity as well as our estimated selectivity. We require a measure for this degradation that allows us to qualify the effect of generalizations. Such a measure is presented in the next subsection. Since the real selectivity of subscriptions is too hard to calculate (cf. Section 4) and thus its degradation $\Delta(s_x, s_y)$, we utilize its estimated counterpart to derive an estimated degradation $\Delta^{\approx}(s_x, s_y)$.

5.1 Selectivity Degradation

There are two options to describe the degradation $\Delta^{\approx}(s_x, s_y)$ of our estimated selectivity between an original subscription s_x and a generalized one s_y:

1. Absolute selectivity degradation
2. Relative selectivity degradation

Here we focus on the absolute degradation because it describes the real influence of generalizations on network traffic. Our absolute measure describes the change in selectivity as the maximal difference between the components of our estimation:

$$\begin{aligned}\Delta^{\approx}(s_x, s_y) \;=\; & \max(sel^{min}(s_y) - sel^{min}(s_x),\\ & sel^{avg}(s_y) - sel^{avg}(s_x),\\ & sel^{max}(s_y) - sel^{max}(s_x))\end{aligned}$$

Its advantage compared to a relative measure is that degradation is expressed by its real change, i.e., $\Delta^{\approx}(s_x, s_y)$ comprises the expected additional number of event messages fulfilling the generalized subscription s_y compared to s_x.

A relative measure would, e.g., handle a decrease in selectivity in a leaf node p_x when generalizing to p_y from $sel(p_x) = 0.4$ to $sel(p_y) = 0.8$ in the same way as a decrease from $sel(p_x) = 0.04$ to $sel(p_y) = 0.08$. Thus, it does not reflect the real influence of generalization on network traffic.

5.2 Weak Points of Selectivity Degradation

We are aware about the difference between the estimated $\Delta^{\approx}(s_x, s_y)$ and the real degradation $\Delta(s_x, s_y)$.

Our estimated measure only describes the degradation in the minimal possible, average and maximal possible selectivity. The real selectivity might change more or less depending on dependencies among attributes. When generalizing a subscription s_x to s_y the worst case real degradation is

$$\Delta(s_x, s_y) \;=\; sel^{max}(s_y) - sel^{min}(s_x)$$

However, calculating the real degradation would be costly in both computational and memory resources. In publish/subscribe systems these resources are very scarce due to large subscription numbers to handle and high frequencies of incoming event messages.

Our generalization approach with its estimated selectivity degradation has still several advantages compared to existing covering and perfect merging techniques, e.g., (Mühl & Fiege 2001): Our approach might not find the optimal generalization in respect to real selectivity degradation, but it is always decreasing the size of index structures in brokers. Covering and perfect merging target a reduction of subscription numbers, which heavily depends on registered subscriptions and is not possible in all cases. Our approach works for all subscriptions, its optimization potential is merely depending on actual dependencies among attributes in event messages.

5.3 Automatic Pruning

Automatic pruning is done in broker components whenever subscriptions forwarded from neighbor brokers require too many resources. In order to execute subscription updates according to estimated degradations, we utilize a priority queue storing $(\Delta^{\approx}(s_x, s_y), s_x)$ tuples. For each incoming subscription s_x we calculate the best pruning leading to s_y. Note that we only need to calculate $\Delta^{\approx}(s_x, s_y)$ and not to determine s_y.

To perform pruning in case of exhausted resources, we generally

1. Extract the first element containing subscription s_x out of our priority queue
2. Perform the best pruning of s_x leading to s_y
3. Remove s_x from index structures
4. Insert s_y into index structures
5. Insert $(\Delta^{\approx}(s_y, s_z), s_y)$ into the priority queue, whereas s_z states the best pruning of s_y

This process is executed as long as enough memory resources have been freed[2].

The selectivity $sel(p_x)$ of predicates p_x might change over time. To incorporate this changing into our pruning process, we have to compare the actual estimated degradation of a pruning operation of subscription s_x to the value stored in the priority queue. If it has changed to a large extend (or is not near the minimum anymore), we can skip pruning s_x and reinsert s_x into the queue associated with the newly calculated degradation.

In case of shared predicates p_x our pruning approach is likely to remove p_x in all subscriptions containing this predicate within a short time. This is due to the general tendency of removing general before more selective predicates. Thus, all of these little selective predicates should be removed early in the pruning process and also relieve predicate indexes.

5.4 Automatic Replacement

Automatic replacement involves the determination of coverings among predicates. Generally, we can replace predicates p_x by all covering predicates p_y. The less selective p_y (chosen to replace p_x), the more our estimated selectivity degradation changes.

However, to effectively decrease memory usage, we need to remove a predicate from predicate index structures[3]. Otherwise, we do not save memory resources because, in contrast to pruning, subscriptions retain their complexity. Thus, the size of subscription index structures remains the same.

Since predicates p_x might be shared among subscriptions, a removal of p_x from predicate index structures is only possible if all subscriptions involving p_x replace its occurrence with p_y. This involves computations of selectivity degradation for all subscriptions involving p_x in order to find the best overall replacement option (thatwhich should be performed first).

This property shows that the computational effort required for predicate replacement is much higher than the one required for subscription tree pruning. Additionally, pruning subscription trees leads to releasing more memory resources. Hence, subscription tree pruning should be preferred over predicate replacement as long as it does not degrade selectivity to a large extend.

6 Experiments and Evaluation

In this section, we present an evaluation and analysis of the automatic subscription tree pruning approach presented in Section 5.3. We focus on this generalization method due to its advantages compared to predicate replacement as shown in Section 5.4.

In the next subsection, we present our experimental setup. We show and analyze our experimental results in Section 6.2 and Section 6.3, respectively.

[2]We should execute Step 3 and Step 4 in batch to allow for a more efficient pruning.

[3]We might be able to save memory in subscription indexes due to implementation-specific memory overhead. Generally, we would remove one entry in the predicate subscription association table for a covered predicate, but reinsert one for the covering predicate.

6.1 Experimental Setup

In several application areas requiring publish/subscribe mechanisms, e.g., healthcare (Jung & Hinze 2005) and electronic commerce (Cilia & Buchmann 2002), more expressive than purely conjunctive subscription languages are required. In our experiments we focus on the popular application of online auctions, which particularly need active functionalities for an efficient dissemination of process-related information (Cilia & Buchmann 2002).

6.1.1 Event Messages

To obtain realistic data for our experiments we have analyzed auctions of fiction books offered on eBay[4] on 8 July 2005. Our analysis focused on attributes shown in Table 1.

Table 1: Overview of attributes for book auctions

Attribute	Example	Values
`Category`	Fantasy	22
`Format`	Hardcover	4
`Special Attribute`	Signed	2
`Condition`	New, used	2
`Ending Within`	1 hour	0 sec . . . 10 days
`Price`	\$0.99	\$0.01 . . . \$1000.00
`Buy It Now`	Yes, no	2
`Bids`	1	0 . . . 100

We have been able to determine the exact number of books for all combinations of `Category`[5], `Format`, `Special Attribute` and `Condition`. Furthermore, we extracted the number of books in all categories for 2 values of `Buy It Now`, 15 ranges of `Price` and 16 ranges of `Bids` (based on the search functionalities offered by eBay). For actual values in these two ranges we assume a uniform distribution, e.g., prices of all fantasy books between \$5.00 and \$6.00 (approximately 7% of all fantasy books) are uniformly distributed in this range.

For prices and bids we compared the distribution of completed and active auctions and realized only minor differences[6]. Thus, we used the distribution derived from active listings in our experiments. For the attribute `Ending Within` we assume a random distribution between 1 minute and 10 days.

We further assume 5 times less authors than books and 10% of all authors have published books in more than one category. The probability of multiple book titles is assumed as 1%. We expect authors and book titles to be given correctly in event messages (e.g., as achievable by utilizing a book database when offering items). We also experimented with other assumptions leading to similar results as presented later.

6.1.2 Subscriptions

Subscriptions in our online auction scenario potentially cover a wide range of user interests. In our evaluation we assume three different subscription classes:

[4]http://www.ebay.com/

[5]We only looked at the first level of categories.

[6]Slightly increased bids and prices in completed auctions.

Subscription class 1. Users are interested in a certain book title. According to the condition (new, used) of the copy of the book, they want to pay a different price. To avoid unnecessary notifications, users want to be notified one day before the end of the auction.

Subscription class 2. Again, users are interested in a certain book title and want to be notified one day before an auction ends. The difference to subscription class 1 is that users further distinguish between different formats, i.e., hardcover and softcover.

Subscription class 3. A collector is interested in books of a certain category, but also of a particular author. He wants to be notified one hour before the end of an auction offering a signed book copy without any bids. Furthermore, he wants notifications about signed books conforming his interests if they are Buy It Now items.

Similarly to event messages, we assume that authors are given correctly in subscriptions. To model subscriptions involving only parts of a book title, we reduce the number of possible titles and assume 100 times less titles than active auction items. We also experimented with other assumptions leading to similar results as presented later.

6.2 Experimental Results

In our experiments we analyze the interconnection between memory usage and network traffic when performing subscription tree pruning. To describe memory requirements, we use the measure of the total number of predicates registered with the system.

We present this measure in a relative manner, i.e., we show the portion of predicates compared to the original situation without applying subscription generalization. Analyzing the number of predicates allows us to directly derive the behavior of the total memory requirements for subscription index structures: For each predicate we can remove one entry from the predicate subscription table. Furthermore, subscription trees do not store removed predicates anymore[7]. Thus, memory requirements for subscription index structures at least decrease in the same manner as predicate numbers.

We neglect the memory for predicate index structures in our analysis due to their high dependency on utilized data types, supported operators and the variety of implementation variants.

For the description of network traffic we utilize the total selectivity of registered subscriptions. This measure directly implies the number of matching events, which in turn implies the network traffic created in event routing.

We separately analyze the three types of subscriptions described in Section 6.1.2 by randomly creating subscriptions conforming to the respective structure. Additionally, we present a setting with randomly chosen subscriptions out of all three classes.

Our results are presented in Figure 6 to Figure 9. Abscissae show the portion of performed pruning operations. Thereby the maximal possible pruning (1.0) describes the case that a further pruning removes a complete subscription, i.e., a subscription either contains of only one predicate or of a disjunction.

Left ordinates in the figures describe the relative decrease in predicates compared to the original situation without any pruning (0.0); right ordinates show the total selectivity of registered subscriptions.

[7]Subscription trees are actually reduced even more, since inner nodes might be deleted when removing predicates.

In our experiments we used 10, 000 subscriptions and created 1, 000, 000 event messages conforming to the derived distribution in online book auctions (Section 6.1.1). The determination of initial selectivities was based on 100, 000 created event messages.

We do not show individual measuring points in our figures due the large number of performed pruning operations (30, 000 to almost 84, 000 in our different settings) leading to the same amount of single measurements.

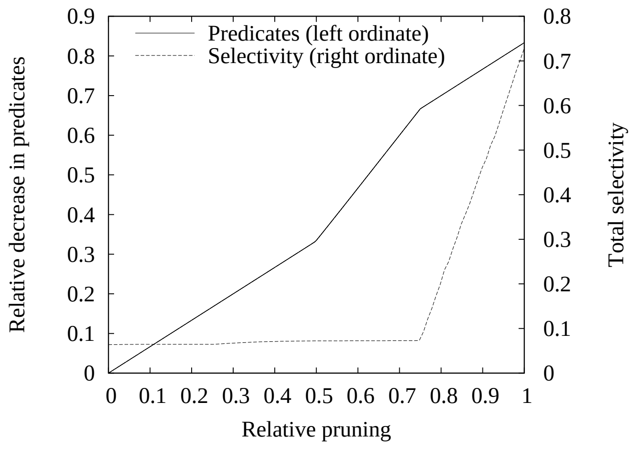


Figure 6: Subscription class 1 - influence of pruning

The behavior of subscription class 1 is given in Figure 6. Looking at the total selectivity of predicates, we realize a slight increase followed by a sharp bend and a fast increase. The bend occurs when approximately 75% of possible pruning operations have been performed (40, 000 in total). At this point, the total selectivity of registered subscriptions has changed by 0.009; relative to their original selectivity (0.065) this is an increase by 14.3%.

Looking at memory requirements, we realize an always increasing behavior. Different gradients result out of pruning different subtrees. Due to the same pattern in subscriptions, similar subtrees are pruned one after the other before proceeding with another one. When reaching the sharp bend, nearly 77% of the predicates, i.e., memory for subscription indexes, have been removed. Thus, a reduction of subscription indexes to 23% of their original size has resulted in a relative increase of selectivity by only 14.3%.

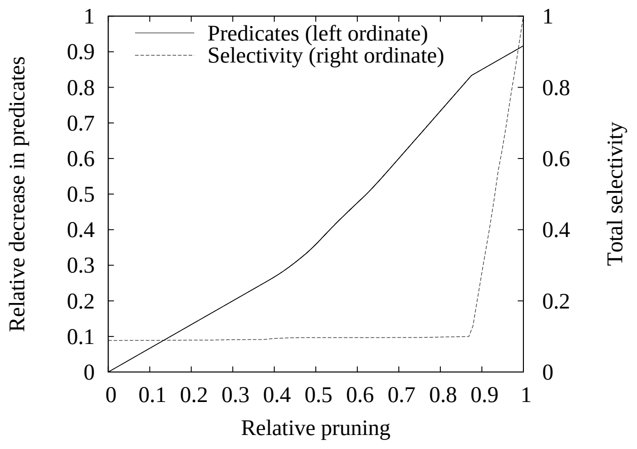


Figure 7: Subscription class 2 - influence of pruning

The behavior of subscription class 2 is depicted in Figure 7. We realize a similar behavior as for subscription class 1: The sharp bend occurs after nearly 88% of performed pruning operations (80, 000 in total). At this point selectivity has increased by 0.012 (13.3%). Memory requirements of subscriptions indexes could be reduced to 17% of their original size.

Compared to subscription class 1, the sharp bend in the selectivity curve occurs after a larger amount of pruning operations has been performed. This directly results in greater savings of main memory resources compared to subscription class 1 before selectivity decreases sharply.

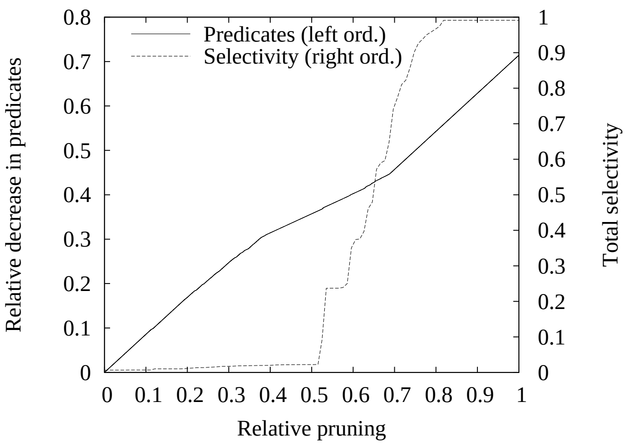


Figure 8: Subscription class 3 - influence of pruning

Subscription class 3 shows a faster decreasing total selectivity than the two previous classes of subscriptions. This is depicted in Figure 8: The sharp bend in selectivity occurs after performing almost 53% of pruning operations (30, 000 in total). Up to this point, selectivity increased by 0.016; memory for subscription indexes could be reduced by 37%.

From data in Figure 8, we observe that the gradient of the selectivity curve increases in steps. These steps result from the inaccuracy of our selectivity measure in combination with the real distribution of event messages: A small amount of pruning operations abruptly increases total selectivity to a large extend (large gradients in curve). Succeeding pruning operations stick to the predicted small decrease in selectivity until the next step occurs.

In this particular class of subscriptions, the reason for this effect is the uneven distribution of signed copies among different categories of books.

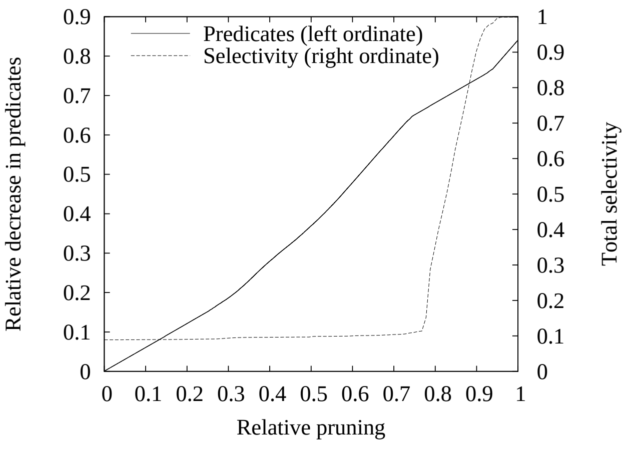


Figure 9: All classes - influence of pruning

In Figure 9 we illustrate the behavior of selectivity and memory usage in a setting with randomly created subscriptions conforming to our three subscription classes. There the sharp bend in the selectivity curve occurs after performing 77% of possible pruning operations (approx. 50, 000 in total). Up to this point, selectivity increases by 0.026 (29%). We realize a strong decrease in memory requirements: subscription indexes could be reduced to 34% of their size.

These results of random subscriptions behave better than expected when considering the results of the three single runs (Figure 6 to Figure 8). This is due to the larger number of pruning operations that are possible for subscriptions of class 1 and class 2 compared to subscriptions of class 3 .

6.3 Discussion of Experimental Results

An overview of our results is given in Table 2. We present our four settings in columns 2 to 5 of the table; rows show six parameters we could derive:

- Overall possible number of pruning operations
- Original selectivity of registered subscriptions
- Relative number of pruning operations at sharp bend compared to possible pruning operations
- Relative increase in selectivity at sharp bend compared to original selectivity
- Total selectivity increase at sharp bend compared to original selectivity
- Relative decrease in predicates at sharp bend compared to original number of predicates

Table 2: Overview of results for our four test settings (the last two parameters describe changes in selectivity and memory at sharp bends)

Parameter	Class 1	Class 2	Class 3	All
Number prunings	40, 000	80, 000	30, 000	50, 117
Original selectivity	0.065	0.089	0.006	0.089
Rel. prunings	0.75	0.875	0.525	0.771
Rel. sel. increase	0.143	0.133	2.589	0.29
Total sel. increase	0.009	0.012	0.016	0.026
Rel. predicates	0.667	0.833	0.368	0.663

Our results show that subscription pruning is an effective way of decreasing the memory required for subscription indexes without increasing the selectivity of subscriptions to a large extend: For random subscriptions in our book auction application, we could reduce subscription indexes by 66% (34% of their original size) with increasing their selectivity by only 29%.

The applied pruning algorithm is straightforward and does solely depend on selectivity of predicates. This makes our optimization method efficient and does not require a large amount of memory.

The efficacy of subscription pruning depends on the structure of subscriptions and the selectivity of their predicates. Especially in cases of combining highly selective and more general predicates in subscriptions, subscription pruning leads to very good results. That is, we can remove various predicates without largely decreasing overall selectivity (which implies network load). Generally, little selective predicates are pruned early without affecting overall selectivity to a large extend. This especially holds if they are located in higher levels of subscription trees. However, the effect of pruning also depends on the structure of subscriptions, i.e., the usage of operators.

Consequently, subscriptions of class 1 and class 2 show better results than subscriptions of class 3. This is because predicates regarding `Ending Within` and `Title` are quite restrictive. Most of the other parts of subscription trees can be pruned without affecting selectivity to a large extend. Subscriptions of class 2 contain more little selective predicates than those of other classes leading to relatively more pruning operations before the sharp bend in selectivity.

For subscriptions mainly involving little selective predicates (e.g., those of class 3), subscription tree pruning can be used, too. Then, the correlation between saved memory resources and increased network usage is not that beneficial as in the former case, but pruning still results in memory requirements decreasing by 37% before strongly affecting selectivity. A selectivity of 1.0 is reached in this case after all possible prunings, because remaining subscription trees query for category that is a relatively general predicate.

We also ran experiments using different settings than presented in Section 6.1. They resulted in other magnitudes of selectivity but similarly developing curves. Especially the assumption of high selectivities for book titles results in increased selectivity.

Several application scenarios normally require subscriptions involving both highly selective and relatively general predicates: Subscriptions in our auction setting involve predicates regarding `Author` or `Title` (high selectivity) but also predicates on `Bids`, `Buy It Now` or `Format` (low selectivity).

Healthcare applications often require notifications in case of critical circumstances, e.g., abnormal blood pressure parameters or, more general, emergencies in intensive care units. These circumstances occur rarely, i.e., specifying predicates are highly selective. Other predicates, e.g., describing identifiers of monitored patients or names of medical conditions, are more general.

We can also find the same pattern in subscriptions for facility management purposes, e.g., when monitoring buildings to detect burglaries. Such events happen rarely (implying highly selective predicates describing, e.g., breakage of glass) whereas ordinary measurements from sensors arrive periodically and are leading to a low selectivity of predicates specifying, e.g., identifiers of certain buildings.

This shows that the structure of subscriptions in various application areas beneficially influences the effect of our subscription generalization approach. Consequently, subscription generalization is a valuable mechanism to increase scalability and efficiency in distributed publish/subscribe systems.

7 Conclusions and Future Work

In this paper we have proposed a novel routing optimization approach for distributed publish/subscribe systems. Our approach, subscription generalization, works on arbitrary Boolean subscriptions in combination with the well-known distribution scheme subscription forwarding. Subscription generalization aims at decreasing the complexity of subscriptions and thus at reducing the memory requirements in filtering broker components. In turn, the selectivity of subscriptions is decreased. In contrast to previous approaches, our proposal works on all kinds of registered subscriptions independent of their similarity and operators used in subscriptions.

We have presented two particular subscription generalization methods: pruning subscription trees and predicate replacement. Our pruning option relieves more memory resources than replacement whereas predicate replacement affects selectivity less than pruning. For subscription pruning, we have proposed an algorithm automatically determining the order of pruning operations based on selectivities.

In order to calculate selectivities of subscriptions, we proposed a simple estimation approach focussing on the minimal, maximal and expected average selectivity of subscriptions. Our estimation is easily computable and requires little additional memory, which is an important quality criteria in publish/subscribe systems due to the large number of subscriptions.

To evaluate subscription generalization, we ran a series of experiments and evaluated our results: In an online auction scenario, subscription tree pruning is an effective way to decrease memory usage in broker components. For a typical set of subscriptions for online auctions, we could decrease memory requirements by 66% while increasing selectivity (and thus network traffic) by only 29%.

Subscription generalization leads to particularly beneficial results when combining highly selective and more general predicates in subscriptions. We can find subscriptions conforming these criteria in several applications, e.g., e-commerce, healthcare and facility management. Thus, subscription generalization is a valuable mechanism to increase scalability and efficiency in distributed publish/subscribe systems.

In the future we plan to integrate subscription generalization as routing optimization in a distributed publish/subscribe service. Then, we want to run an advanced series of experiments directly evaluating network traffic, memory requirements and efficiency.

References

Bittner, S. & Hinze, A. (2004), Classification and Analysis of Distributed Event Filtering Algorithms, *in* 'Proceedings of the 12th International Conference on Cooperative Information Systems', Agia Napa, Cyprus, pp. 301–318.

Bittner, S. & Hinze, A. (2005*a*), Investigating the Memory Requirements for Publish/Subscribe Filtering Algorithms, Technical Report 03/2005, Computer Science Department, University of Waikato.

Bittner, S. & Hinze, A. (2005*b*), On the Benefits of Non-Canonical Filtering in Publish/Subscribe Systems, *in* 'Proc. of the 25th IEEE International Conference on Distributed Computing Systems Workshops', USA, pp. 451–457.

Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (2001), 'Design and Evaluation of a Wide-Area Event Notification Service', *ACM Transactions on Computer Systems (TOCS)* **19**(3), 332–383.

Carzaniga, A., Rutherford, M. J. & Wolf, A. L. (2004), A Routing Scheme for Content-Based Networking, *in* 'Proc. of the 23rd IEEE Conference on Computer Communications', China.

Chand, R. & Felber, P. A. (2003), A Scalable Protocol for Content-Based Routing in Overlay Networks, *in* 'Proceedings of the Second IEEE International Symposium on Network Computing and Applications', Cambridge, USA, pp. 123–130.

Chen, Z., Koudas, N., Korn, F. & Muthukrishnan, S. (2000), Selectively Estimation For Boolean Queries, *in* 'Proc. of the 19th Symp. on Principles of Database Systems', USA, pp. 216–225.

Cilia, M. & Buchmann, A. P. (2002), 'An Active Functionality Service For E-Business Applications', *ACM SIGMOD Record, Special Issue on Data Management Issues in Electronic Commerce* **31**(1), 24–30.

Crespo, A., Buyukkokten, O. & Garcia-Molina, H. (2003), 'Query Merging: Improving Query Subscription Processing in a Multicast Environment', *IEEE Transactions on Knowledge and Data Engineering* **15**(1), 174–191.

Fabret, F., Jacobsen, A., Llirbat, F., Pereira, J., Ross, K. & Shasha, D. (2001), Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems, *in* 'Proc. of the 2001 ACM SIGMOD International Conference on Management of Data', USA, pp. 115–126.

Guimarães, M. & Rodrigues, L. (2003), A Genetic Algorithm for Multicast Mapping in Publish-Subscribe Systems, *in* 'Proc. of the 2nd IEEE International Symposium on Network Computing and Applications', USA, pp. 67–74.

Halevy, A. Y. (2000), 'Theory of Answering Queries Using Views', *ACM Special Interest Group on Management of Data Record* **29**(4), 40–47.

Hanson, E. N., Chaabouni, M., Kim, C.-H. & Wang, Y.-W. (1990), A Predicate Matching Algorithm for Database Rule Systems, *in* 'Proc. of the 1990 ACM SIGMOD International Conference on Management of Data', USA, pp. 271–280.

Hinze, A. (2003), A-MEDIAS: Concept and Design of an Adaptive Integrating Event Notification Service, PhD thesis, Freie Universität Berlin, Institute of Computer Science.

Jung, D. & Hinze, A. (2005), A Mobile Alerting System for the Support of Patients with Chronic Conditions, *in* 'Proc. of the 1st Euro Conference on Mobile Government', UK, pp. 264–274.

Mathieson, I., Dance, S., Padgham, L., Gorman, M. & Winikoff, M. (2004), An Open Meteorological Alerting System: Issues and Solutions, *in* 'Proc. of the 27th Australasian Computer Science Conference', Dunedin, New Zealand, pp. 351–358.

Mühl, G. (2001), Generic Constraints for Content-Based Publish/Subscribe Systems, *in* 'Proc. of the 6th International Conference on Cooperative Information Systems', Italy, pp. 211–225.

Mühl, G. (2002), Large-Scale Content-Based Publish/Subscribe Systems, PhD thesis, Technische Universität Darmstadt.

Mühl, G. & Fiege, L. (2001), 'Supporting Covering and Merging in Content-Based Publish/Subscribe Systems: Beyond Name/Value Pairs', *IEEE Distributed Systems Online* **2**(7).

Pereira, J., Fabret, F., Llirbat, F. & Shasha, D. (2000), Efficient Matching for Web-Based Publish/Subscribe Systems, *in* 'Proceedings of the 7th International Conference on Cooperative Information Systems', Eilat, Israel, pp. 162–173.

Poosala, V. & Ioannidis, Y. (1997), Selectivity Estimation Without the Attribute Value Independence Assumption, *in* 'Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB)', Athens, Greece, pp. 486–495.

Wang, Y.-M., Qiu, L., Verbowski, C., Achlioptas, D., Das, G. & Larson, P. (2004), 'Summary-based Routing for Content-based Event Distribution Networks', *ACM SIGCOMM Computer Communication Review* **34**(5), 59–74.

Yan, T. W. & García-Molina, H. (1994), 'Index Structures for Selective Dissemination of Information Under the Boolean Model', *ACM Transactions on Database Systems (TODS)* **19**(2), 332–364.

The Challenge of Creating Cooperating Mobile Services: Experiences and Lessons Learned

Annika Hinze[1] **George Buchanan**[2]

[1]Department of Computer Science, University of Waikato, New Zealand
hinze@cs.waikato.ac.nz
[2]UCL Interaction Centre, London, United Kingdom
g.buchanan@cs.ucl.ac.uk

Abstract

In this paper, we present a mobile infrastructure for cooperating information services. This infrastructure is demonstrated through the example of a Tourist Information Provider (TIP) system. TIP delivers context-sensitive information from a variety of services to the user. The underlying communication is event-based to support continually changing information.

We demonstrate two examples of the use of context-sensitive services in TIP: first, the presentation of sight information using a Zoomable Map service which links into a detailed information service; second, the exploitation of contextual information to deliver targeted recommendations to the user. Through these examples, we demonstrate the requirements for mobile multi-service systems that support flexible cooperating services.

1 Introduction

Complex information systems are increasingly required to support the delivery of information to mobile devices. Studies of these devices in use have demonstrated that the information displayed to the user needs to be limited in size and focussed in content (Buchanan & Jones 2000). Furthermore, the presented information is often dynamic – even changing continuously. Event-based communication provides strong support for selecting relevant information for dynamic information delivery. Therefore, an event-based system meets many requirements for mobile information systems.

In this paper, we present a mobile infrastructure for cooperating information services. The infrastructure uses an event-based communication layer to support continually changing information. This new infrastructure extends and enriches our earlier core system by providing cooperating services, and additional information and features to the mobile user. The services had to be modular and loosely coupled to enable users to choose different services depending on their travel context (e.g., displaying data on a map or in textual form). This novel approach differs from existing mobile information systems in its support for open, componentised and cooperating information services and its use of an event-based communications that better support the supply of up-to-date data in a fluid information environment.

The work reported here builds on our first-generation stationary TIP 1.0 system, which was reported in (Hinze & Voisard 2003, Hinze, Loeffler & Voisard 2004). Previous work focussed on the interplay of different event/information sources and the event-based information delivery. The system evolved through several versions undergoing considerable changes; we refer to the new TIP version that is introduced here as TIP 2.9.

This paper is structured as follows: In Section 2, we demonstrate the core TIP system in use to highlight the functional demands and constraints for the design. We identify the challenges of cooperating modular services, and define the focus of this paper. In Section 3, we introduce our new conceptual design for TIP 2.9. We discuss in detail two services cooperating with the TIP core system: the map service and the recommendation services incorporated into the second-generation TIP. This will be followed by a technical examination of TIP 2.9, considering the software's implementation details (Section 4). Subsequently, we discuss the issues identified and lessons learned for inter-operating and cooperating services in a mobile context (Section 5). In Section 6, we review previous work in the field, and clarify the contributions of the new TIP. The paper closes with a summary and discussion of future work (Section 7).

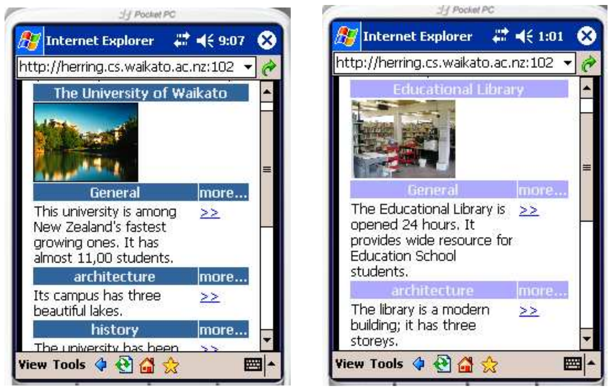
2 TIP: The Core System

In this section, we briefly re-visit the TIP 1.0 system in use (for more details on the early versions of TIP see (Hinze et al. 2004, Hinze & Voisard 2003)). We demonstrate the general principles of the TIP system first in a usage scenario and then as conceptual architecture. We conclude the section by highlighting the challenges we faced when integrating cooperating services into the TIP system.

2.1 TIP Usage Scenario

For clarity, we will explore the simple example of a visitor Peter coming to New Zealand. Peter is a visiting researcher from London. He has his own TIP-enabled PDA, and uses it as an interactive guide to assist his choice of where to visit. Peter is keen to visit some historic buildings in New Zealand and is generally interested in architecture. In our example scenario, Peter is visiting the campus of the University of Waikato.

Peter has set up his profiles in TIP: choosing the sight types of museums and campus buildings (in his *sight profile*) with information topics of 'history' and 'architecture' (in his *topic profile*). The TIP system will prioritize the display of sights of interest relevant to his interests expressed in the sight profile. TIP will only give general introductory information and


Internet Explorer 9:07
http://herring.cs.waikato.ac.nz:102
The University of Waikato
General more...
This university is among New Zealand's fastest growing ones. It has almost 11,00 students. >>
architecture more...
Its campus has three beautiful lakes. >>
history more...
The university has been >>

Internet Explorer 1:01
http://herring.cs.waikato.ac.nz:102
Educational Library
General more...
The Educational Library is opened 24 hours. It provides wide resource for Education School students. >>
architecture more...
The library is a modern building; it has three storeys. >>

(a) current location, with darker colour scheme

(b) distant location, with lighter colour scheme

Figure 1: TIP information delivery based on the user's location, interests, and travel history: (a) Delivering information about user's current location, (b) Browsing for information about a distant location

information pertinent to the topics that Peter selected in his topic profile.

When standing at the entrance to the University, Peter's TIP display gives general information about the University. The display is shown in the screenshot in Figure 1(a), which features our new mobile interface of TIP. Peter can also browse for places that he may want to visit but that are not directly at his current location (see screenshot in Figure 1(b)). Peter can obtain further travel possibilities, either further away or on different themes by interacting with his TIP display. On revisiting places Peter has been to before, the system displays the latest information that was given to Peter on his last visit.

2.2 TIP Core Concepts

TIP's information delivery is based on the user's context: their location, personal profile describing interest in (semantic) sight groups and topics, and the user's travel history. The system also considers a sight's context, such as its location and its membership in predefined semantic groups of sights.

The TIP core system is implemented using an event-based infrastructure combined with a location-based service. See Figure 2 for a conceptual architecture of the TIP core system. The heart of the system is the filter engine cooperating with the location engine. The filter engine selects the appropriate information from the different source databases based on the user and sight context. Changes in the user's location are transmitted to the TIP server (Steps ① and ②), where they are treated as events that have to be filtered. For the filtering, the sight context (Step ③) and the user context (Step ④) are taken into account. The location engine provides geospatial functions, such as geocoding, reverse geocoding, and proximity search (Step ⑥). For places that are currently of interest to the user, the system delivers sight-related information (Step ⑦).

In addition, notification about scheduled events, such as opening hours of museums and theater program information are offered by the system. Notifications about external events may also be given to the users, e.g., about changed starting time for a theater performance. Note that these kinds of events are handled similar to the location events but follow different characteristics: scheduled events occur infrequently compared to user-driven location events.

Information about sights and other spatial data

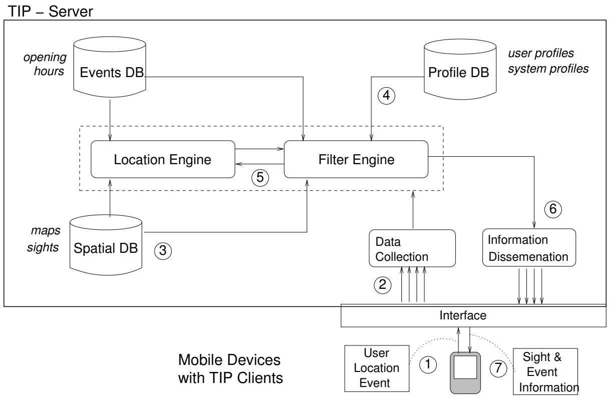


Figure 2: Conceptual architecture of the TIP core system: Component interactions

are stored in the *spatial database*; event-related information is stored in the *event database*. The operations of the filter engine are controlled by *user profiles* and *system profiles* (stored in the *profile database*). TIP uses information about users' preferences and user-sight–relations (such as delivered information and visited sights) for it's information delivery. System profiles enable a flexible integration of different applications; this aspect of the system is described in more detail in Section 3.1.

The system is implemented as a client-server architecture, supporting desktop clients (as reported in earlier publications) as well as mobile clients on a hand-held device with appropriate interfaces (presented in this paper for the first time).

2.3 Challenges of Cooperating Mobile Services

As described above, TIP delivers context-sensitive information to mobile users. Having developed the TIP core, we discovered interesting additional features with which we wished to enhance the system. Considering the traveller application, one may want to deliver and display the sight related information in different forms. Options are, for example, on map displays (Jones, Jones, Marsden, Patel & Cockburn 2005) or using audio guides (Warren, Jones, Jones & Bainbridge 2005). Often, travellers would also like to interact with the system to plan their journey or be inspired to visit new places. One could envision using travel planners as pre-travel aids and recommendations as interactive support during journeys (Hinze & Junmanee 2005).

The challenges we identified for these enhancements are threefold; the additional features should be:

1. *modular services* that can be used in addition to the core system, allowing the users to use different services for similar purposes interchangeably (e.g., for displaying guidance information using maps or textual representation).

2. *cooperating services* that exchange context data and information for the benefit of the system's user.

3. *mobile services* that can be used on typical hand-held devices; preferably with little or no installation and maintenance overhead for the user.

In this paper, we demonstrate two examples of the use of context-sensitive services in TIP: first, the presentation of sight information using a Zoomable Map

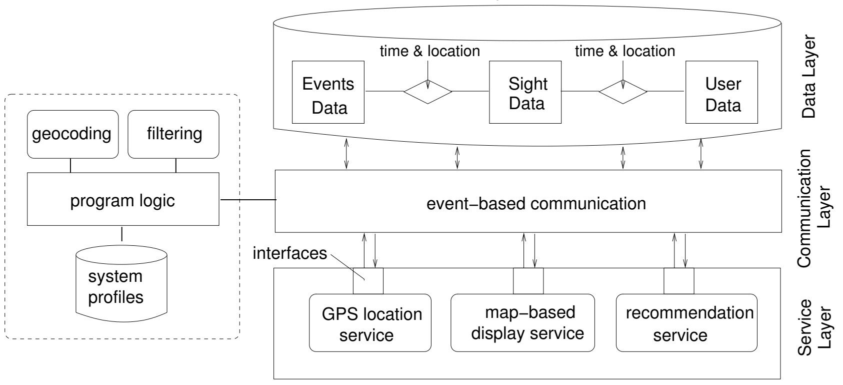


Figure 3: Extended conceptual design of the TIP system: A mobile infrastructure for cooperating information services

service which links into a detailed information service; second, the exploitation of contextual information to deliver targeted recommendations to the user. We will analyse and demonstrate the requirements for mobile multi-service systems, discuss the benefits of the TIP approach and the lessons learned.

3 TIP Service Architecture – Design

In this Section, we first describe the conceptual design of the extended TIP system (in Section 3.1). Then, we illustrate the functionality and interactions of two services cooperating with TIP: a map service (in Section 3.2) and a recommendation service (in Section 3.3).

3.1 TIP Extended Conceptual Design

To support modular cooperating information services in a mobile TIP environment, we developed a layered conceptual architecture as shown in Figure 3.

As before, the heart of TIP still lies in the event filter engine (for *filtering*) cooperating with the location engine (for *geocoding*) (shown in the dashed area on the left in Figure 3). The *system profiles* determine the interaction patterns and the functionality of the system. Supporting variable system profiles will enable a flexible integration of different applications – the TIP infrastructure could then be used for a variety of differing applications, such as a museums guide, an electronic learning environment, or a mobile support for patients with chronic conditions (as suggested in (Jung & Hinze 2005)). Details about the design of system profiles can be found in (Hinze & Voisard 2003).

For the *Communication Layer*, the inherent[1] event-based communication of the core TIP system design has to be modularized and extended into an event-based communication infrastructure. All communication is based on event messages that are forwarded to the respective components or services. The communication is either direct and unfiltered (e.g., forwarding of parameters), or controlled by the filtering logic (e.g., selection of appropriate information or service). Note that the event-based communication infrastructure will have support frequent location events from a high number of users.[2]

[1]The origin and core of the filter engine is an event notification service that is employed here to filter location-dependent events.

[2]We are currently evaluating these requirements of performance and scalability for the event-based communication. Another issue to consider is security and privacy. Both are currently addressed on the application level as well as on the communication level.

The *Data Layer* holds user-related data as well as data regarding the sights and information about external and scheduled events. User data and event data are related to sight data by (often time-dependent) references in location. For example, users visit sights (i.e., locations) at certain times, which is then recorded in their travel history. The opening hours of a museum are described as scheduled events; similarly the list of performances at a theater.

The *Service Layer* provides interfaces to different types of services. Communication among services as well as communication between the TIP core and the services are channelled via the communication layer. One basic service is a location service, here shown as GPS-based service. Using the layer structure, it will be possible to interchangeably use different location services that provide information about the user's current location: the service implementation and technical details (e.g., using GPS or wireless LAN) are transparent to the other parts of the system.

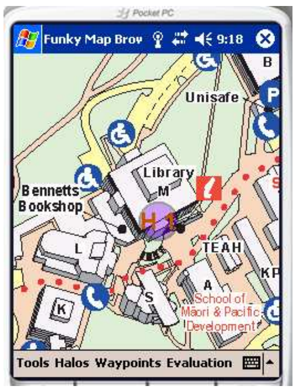
3.2 Cooperating Services: Maps in TIP

For users navigating in physical locations, maps can provide vital information about the surrounding area. On small physical devices, the display of detailed maps within the confines of a limited screen area can be problematic. We wished to assist the user's conception of their own context by supplementing the existing TIP system with an interactive map. We used a third-party Zoomable Map display tool that is tailored for use on small displays (Jones et al. 2005).

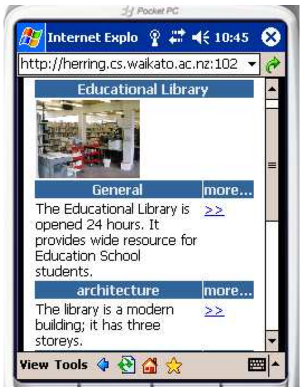
We adapted the map display tool to be used as a service in TIP. An example display of the Zoomable Map interface in TIP is shown in Figure 4(a): the sight at the current location is indicated by a colored circle, which can be seen at the center of the screenshot. The Zoomable Map interface can also support indicators for several locations, which would then appear as rings at the edges of the screen to indicate nearby sights that are just off the visible map (for clarity, we omit this feature in Figure 4(a)).

Here, we will focus upon the conceptual integration of the tourist information service inside TIP and the additional Map service. The core TIP system runs on a web server, whilst the Zoomable Map service for PDAs runs as a thick client on mobile devices themselves.

The TIP display application runs as a separate application (thin client) on the PDA. Referring to our Conceptual Design in Figure 3, the TIP display application is a service for which the map service provides an alternative display service.



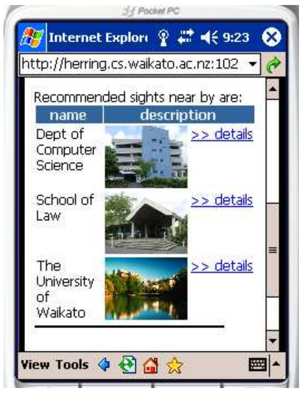
(a) map display for current location



Educational Library

General	more...
The Educational Library is opened 24 hours. It provides wide resource for Education School students.	>>
architecture	**more...**
The library is a modern building; it has three storeys.	>>

(b) TIP sight information for current location (Educational Libraray)



Recommended sights near by are:

name	description
Dept of Computer Science	>> details
School of Law	>> details
The University of Waikato	>> details

(c) recommendations for interesting sights near by

Figure 4: Interplay of services: The central screenshot (b) shows the TIP core interface at the current location of the Educational Library; (a) shows a screenshot of the map interface at the same location; (c) shows a screenshot of the recommendation service interface at this location

Location information on the PDA is obtained through a GPS receiver (running as another service with a thick client on the mobile device), and is delivered via the event-based communication layer to the TIP core system and to other interested services. In this case, the information is passed on to the Zoomable Map application to give the current 'home' location (centering the map).

At the TIP core, the information is processed and corresponding sight information is passed back to the services. This information on sights retrieved from the TIP server needs to be forwarded to the TIP client software on the PDA. In addition, the TIP client then needs to pass its sight/user information onto the sight display module of the Zoomable Map application. The sight display module is responsible for drawing the sight circles over the underlying map. Note that indicating sights on the map depends not only in the current location (obtained from the location service) but also the user's interest (obtained from the TIP system) – only sights that are judged as being of interest for the user are pointed out.

3.3 Cooperating Services: Recommendations in TIP

TIP presents information based on user and sight context, as described in Section 2. In addition, recommendations suggest new sights that he user might be interested in. In the earlier TIP version, the sight recommendations delivered to the users only reacted to the *current context* of the user.

In TIP 2.9, we have supplemented the original presentation with a recommendation component that utilises the user's *known preferences* and the current context of user and sights. User preferences are determined not only based on the user's current profiles (regarding sight groups and topics), but depending on information from previous user sessions: the user's previous context (travel history) and their feedback about the sights they visited. In addition, we use information about other users.

Part of the wider goals of the TIP project is to evaluate the different methods for providing user recommendations. Therefore, our recommender component needed to support modular recommendation services that could be combined, added and removed as required. In addition, the provision of recommendations requires a separation of logic at different levels: As we wished to have a candidate test rig for exploring the different options, a modular approach was required for the services. In the presentation of the results, the service also needed to abstract from any given display method (textual or map-based) and from any particular recommendation algorithm. In this section, we will discuss both the functionality of the recommendation services and considerations for the affected components of the TIP architecture.

As described in Section 2, TIP uses detailed information about sights and users to tailor the presented tourist information. As an example, Figure 4(b) shows tailored information about the Educational Library as it is presented to our visitor Peter, who is interested in campus buildings, architecture and history. To present personalised recommendations, we can use the same base data. In addition, the recommendation service needs to extend the data held about the users: each user's feedback about the visited sights is recorded.

We implemented three different recommendation services RS1–RS3 that utilize different information as basis for the recommendations: (RS1) user preferences and sight context, e.g., interest and proximity, respectively; (RS2) the user's travel history and their feedback; (RS3) feedback and preferences of similar users. For a given situation, the user can interactively choose between the different recommendation services. Figure 4(c) shows a screenshot of the recommendation interface presented to our traveller at the Educational Library; here we used the recommender service RS1.

Our services explored various data combinations as input for recommendations comparatively to explore the viability of each option, and the quality of recommendations that could be achieved in different usage scenarios. The service RS1 can be used immediately after entering the TIP system for the first time. Earlier user feedback is utilized to compute personal and user-specific recommendations in RS2. Information about similar users is helpful to discover new sights a user might not have been aware of (in RS3). Different information, such as proximity or personal interest, is utilized to prioritize the display of material to the traveller. For further details about the design and evaluation of the travel recommendation services, see (Hinze & Junmanee 2005).

As can be seen from these considerations, a strong server-side cooperation with the existing data and core services is necessary. We therefore implemented the recommendation services as server-based services

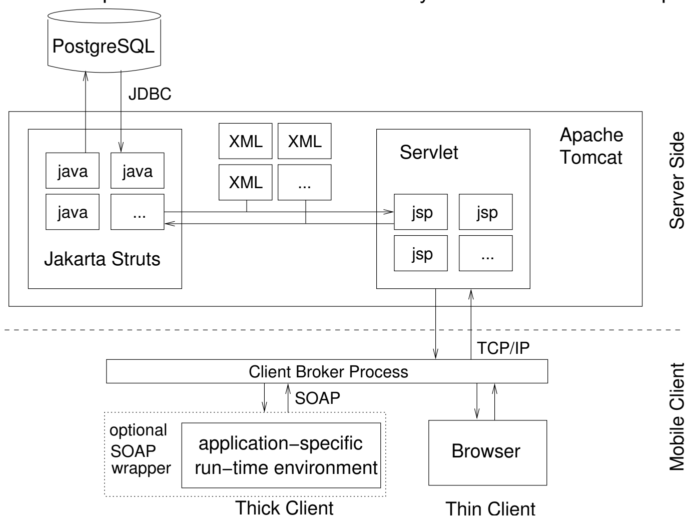


Figure 5: TIP 2.9 Architecture: Implementation details

with thin clients. Details about the implementation of the server–client communication are given in the next section.

4 TIP Service Architecture – Realization

This section discusses the implementation of the TIP architecture. An earlier implementation of TIP 1.0 is described in (Hinze et al. 2004), where we contrast two implementations using a relational database model and a semantic web RDF model, respectively. Here, we focus on the interplay of the TIP core architecture, the mobile clients, and the cooperating services. We first introduce the technical details of the TIP architecture and then discuss the implications of two differently structured services cooperating with TIP.

4.1 Architecture of TIP: Technical Details

The technical details of the TIP architecture are shown in Figure 5; we demonstrate the TIP server and its interaction with both a thick and a thin client, respectively. TIP supports both client architectures, since different services cooperating with TIP might require either thin or thick clients (such as those required for the recommendation services and the map service, respectively).

On the server side, the TIP software has a database back-end using a PostgreSQL database with PostGIS extensions for the geographic data. The server software is implemented using Apache's Jakarta Struts framework as a flexible control layer. Supported by the Struts framework, TIP's architectures is based on the Model-View-Controller (MVC) design paradigm. It supports three separate modules: one for the application model with its data representation and business logic (java files), the second for views that provide data presentation and user input (JSP files and tag library JSTL), and the third for a controller to dispatch requests and control flow (XML files). A servlet manages the execution of the JSP files and their corresponding java files. The Struts framework is also used for the internationalization of TIP (i.e., separate storage of text and application) by supporting different language interfaces (German and English). For more details on the Struts layers of TIP see (Ottlinger 2004).

The TIP core architecture requires only a thin client for the delivery of sight information and the client interfaces for registration and profile definition. TIP provides a desktop interface (for standard browsers) and a mobile interface (standard bowser or TIP browser, depending on the mobile device).

Services cooperating with TIP may require thin or thick clients. Thin clients may use the mobile TIP browser or a web browser for displaying HTML/JavaScript.

Thick clients are application specific; they operate in their own run-time environment. Communication with the TIP server is typically managed via TCP/IP. Services may also interact (cooperate) with each other: Interaction between services is performed via the TIP server, since only the server holds the necessary interface definitions.

For multi-process management on the mobile device, an additional broker process is needed. We experimented with various arrangements of multi-process service management (which we will discuss in Section 4.3). The arrangement shown in Figure 5 seems to overcome most shortcomings. The broker process manages the client services as sub-processes and, thus, emulates task management and allows for inter-process communication.

4.2 Cooperating with the Map Service (Thick Client)

The basic communication with a thick client is complicated by the limitations of the available protocols on the PocketPC platform (or the similar restrictions of alternatives such as PalmOne). Any truly mobile application must be developed within these constraints. For example, even simple technologies such as 'pipes' between processes are not supported by the existing APIs. The new generation of 'smartphones' shares some similarities to PDA systems, but often have even fewer communication options. Naturally, one key challenge in designing and implementing a complete mobile infrastructure for cooperating services is the delivery of the required service functionalities with such limitations.

Furthermore, the system must support service modularity and cooperation both within the framework (e.g., the TIP client) and to services outside the framework (e.g., Zoomable Map). It is critical to also support TIP services which primarily exist within other applications and services (e.g., the sight display module).

In this subsection, we consider the arrangement for cooperation between TIP, the map service and TIP sub-services. Cooperation with other client services is discussed in Section 4.3. Cooperation between the TIP core and the map service is managed via TCP/IP (socket-based) communication, either directly or via the TIP client broker. We planned to use the map service to display sight-related information on maps, such as current position, current sight, and sights near by. The original Zoomable Map software supported positioning of a number of halos (circles to identify locations) by the user, and a zoom-feature in respect to the given halos.

In order to display TIP's information, we needed to extend the Zoomable Map software by an additional service; the sight display module (SDM). The sight display module is planned to support both the geographical positioning of halos and the indication of different semantics (e.g., 'home' and 'other place'). Semantic information may be either displayed explicitly (e.g., a sight name), or communicated to the user through subtle cues such as the use of different shapes or colours. The current implementation only supports the display of visually similar halos at coordinates of sights known to the the core TIP system. The sight display module runs as a separate sub-service of the Zoomable Map (i.e., as a sub-process). This allows for direct communication between the two processes without server interaction, and readily overcomes some difficulties with communication within the PocketPC API. This approach delegates part of the display decisions and context away from the core TIP server, allowing greater interactivity. Processing a long feedback loop via the server to obtain information whilst the user scrolls across the map would be wasteful and potentially error-prone, and may cause delays in providing the user with accurate information.

Clearly, this obliges the 'on-board' storage of some sight information. Handheld devices have sufficient storage to cache simple sight information (e.g., location, name) intelligently within a local context, and the initial download at a location typically consists of only a few seconds. Such data could also be cached in advance. Further detailed information is obtained on a simple on-demand basis. The Zoomable Map system further supports a hybrid thick/thin client approach, wherein the recommendation service (see following subsection) can be viewed through a browser sub-process. Thus, in implementing the Zoomable Map, we have probed thick, thin and hybrid clients, and also some simple caching requirements. All of these operate within the extended TIP framework. Services are mutually aware, and communicate with the underlying TIP databases through abstracted APIs that hide platform-specific and protocol-specific implementation details.

4.3 Cooperating with the Recommender Service (Thin Client)

We consider the communication both between the recommendation service and the server, and in between services. The basic communication of TIP with thin clients or between thin clients is simple and does not differ from interactions and cooperation using a desktop computer. Communication is performed directly via TCP/IP using HTTP.

The cooperation between and management of services that involve mixed clients, e.g., between the map service with its own interface and the recommender services using a browser-based interface, is more challenging. It is not sufficient to have independent client processes running on the mobile device. When using this type of task management, it is not possible to flexibly change between the different interfaces and to support a guided and controlled user interaction with the TIP system. The services would act as independent applications.

We therefore propose to employ a TIP client broker process using the following set-up: Each TIP service runs as a separate process and the independent client broker process is utilized for the management of processes. Thus, the TIP service processes have to be instantiated by the broker service. This allows for flexile management of tasks, e.g., to switch between interfaces. To enable loose coupling, the communication between the broker process and its sub-services should utilize the SOAP protocol, if supported by the respective services. For third party services, SOAP wrappers may have to used; otherwise the service's interface definition has to be directly implemented into the broker interface. This would result in tight coupling of services which is contrary to the desired modularity of TIP services.

5 Discussion: Issues Identified and Lessons Learned

This paper presented a number of difficulties and challenges for creating an event-based communication framework for mobile systems. In this section, we will summarize the experience and lessons learned to date (early impressions of our experiences were given in (Hinze & Buchanan 2005)).

- Firstly, communication protocols themselves are problematic, given the limited range of inter-process communication techniques available between processes running on the same mobile device, and the various options available when communicating between a mobile device and the core (TIP) server. A global framework is obliged to hide such implementation details from the different components of the system, to provide a consistent framework both at the present time and over the changes that one can readily anticipate in the future. Therefore, the TIP framework must abstract over TCP/IP, SOAP, Windows Messaging and other APIs to support seamless communication between service components.

- Secondly, an apparently simple final service provided to the user (e.g., a tourist guide with map, sight data and recommendations) is in fact a composition of a variety of services. Often, these services need to communicate together both within the same machine and between computers, using thick- and thin-client scenarios, and occasionally in hybrid approaches. A sound framework must also support this range of requirements. Unlike a monolithic approach, often seen in existing mobile information systems, a modular, service-oriented approach allows for the exploration of alternatives, e.g., of communication (c.f. Zoomable Map) or implementation (c.f. recommendations). This is important where even fundamental services such as location can be provided in different ways – e.g., GPS or 3G wireless telephony. Furthermore, new services can add entirely new features to the framework without requiring the re-implementation of or changes to existing services.

- The frequent development of stand-alone mobile services has, we believe, resulted in higher development costs for mobile information systems, reducing the rate of new research. Substantial effort is required to provide a basic communication infrastructure. By providing a common framework, this substantial front-loaded implementation cost can be dramatically reduced. The use of a modular, service-oriented approach in the related field of digital libraries (e.g., (Bainbridge & Witten 2004)) has demonstrated the advantages of this approach both practically and for research.

- We have successfully created a framework in which mobile services cooperate. This can already be seen in the communication between the map and sight information services to provide sight location halos in the Zoomable Map. In future, further forms of cooperation and composition are needed. For example, we wish to extend the display module in the Zoomable Map system to support different halo cues (e.g., for sight type or recommendation) and this requires inter-process communications that may in turn require further development of our framework.
- However, implementation details, particularly issues of standardisation, continue to be relevant. For example, in the case of the Map system, different mapping scales and notations are used by different map and information providers, and further services must be introduced to mediate between systems that function in different notational standards.
- With the trend of information systems moving onto mobile devices or supporting mobile clients, the challenges identified in this paper will become more pronounced. Client devices will provide a number of pre-installed services and users will add their own selections. Consequently, we believe that even stronger decoupling and modularization may be needed: A mobile infrastructure for mobile information services needs to flexibly support existing, changing or new services. The next design step in the TIP project will therefore see the completion of re-designing TIP into a Service-Oriented Architecture (SOA) using web services (TIP 3.0).

6 Related Work

TIP is not the first mobile tourist information system. However, it is the first to use a modular, event-based communication architecture.

In the first part of this section, we discuss the existing systems against which TIP can be compared, identify the system requirements recorded in the existing literature, and compare the TIP Service Architecture and existing implementations against these needs. We conclude the section with a discussion of the requirements that we have met, and the outstanding issues for the TIP architecture.

In the second part of this section, we discuss related approaches from other fields, such as web services, agents, ubiquitous computing and context-aware systems.

Mobile Tourist Information Guides From our own studies and based on the pertinent literature, we identify the following requirements for a mobile infrastructure for cooperating services in the area of tourist information. We group the requirements into three groups: architecture, application, modelling. We reported results of an extensive study about modelling requirements in previous work (Hinze & Junmanee 2005, Hinze et al. 2004); here we therefore concentrate on architecture and application. Architectural requirements are support for modular, cooperating, mobile services, mobile services and support for both thin and thick clients (as discussed in Section 2.3) Application requirements are support of personalization, of user location guidance (e.g., maps or music), and of recommendations. We consider only systems that provide location-based sight information (see Table 1).

Most research in the area of electronic guides has focussed on indoor user. Two of the most widely documented systems for outdoor use are CyberGuide and Guide. We additionally included systems that had similar objectives for comparison.

AccessSights (Klante, Krsche & Boll 2004) is a multi-modal location-aware mobile tourist information system that provides tourist information to both normally sighted users and visually impaired people. Both visual display and auditory information is given to users; the system uses loudness to indicate distance between users' current location and attraction spots.

The CATIS (Pashtan, Blattler & Heusser 2003) system is a web-based system that recommends restaurants based on the users preference and travel history. The system also considers the current context such as the location and time of the day as well as means of travel and speed and direction of travel (to determine the restaurants that can be reached within a reasonable time). The system uses an architectures based on web services; it provides different XSLT style sheets to support display on for PC, PDA, and mobile phone.

CRUMPET (Poslad, Laamanen, Malaka, Nick, Buckle & Zipf 2001) is a mobile system that provides personalized and location-aware services to tourists. To interact with the system, a user first provides personal information; the system learns more specific user preferences during the user interactions with the system. CRUMPET provides tourist information according to the user's location. Crumpet uses an architecture based on agents; it provides interfaces for displaying on laptop, PDA, and mobile phone.

CyberGuide (Abowd, Atkeson, Hong, Long, Kooper & Pinkerton 1997) is a mobile system that assists a visitor in a tour of Georgia Tech Lab; it was extended to also support outdoor use. The system mainly focuses on investigating context-sensitive computing so that only limited support for tourist information is provided. The project consists of a family of prototypes with several independent components.

Guide (Cheverst, Mitchell & Davies 2002) is a mobile context-aware tourist guide facilitating visitors while they are travelling the city of Lancaster. The user has access to different functions, such as retrieval web information based on their current location, booking a a restaurant for dinner, and message sending. The Guide system offers a personalised 'Nearby Attractions' page on which it recommends sights that are near by the users current location. The system uses interactive dialogues for personalization.

Gulliver's Genie (O'Hare & O'Grady 2003) is a mobile context-aware service for tourist information. It delivers travel information depending on the users location and context. Gulliver's Genie uses an agent-based approach: It supports the deployment of intelligent agents to flexibly assemble multi-media presentations that are displayed on a PDA. The authors point out that the system is rather demanding in respect to the quality and size of the client device.

The TIP core system (Hinze et al. 2004, Hinze & Voisard 2003) provides rich location-based information depending on user context; support for modular services and flexible client configuration are poor. Due to the web-based approach, the system has only few requirements regarding the client device.

As we see from the comparison presented here that existing systems fall short in respect to the requirements and the aim reported in this paper. The TIP 2.9 system is the first one to address the problem

System	Architecture		Application		
	modular services	thin/thick clients	persona-lization	location guide	recommen-dations
AccesSights	+	–	–	+	–
CATIS	+	o	+	–	+
CRUMPET	–	–	+	?	–
CyberGuide	o	?	–	+	–
Guide	–	–	o	–	+
Gulliver's Genie	o	–	+	+	+
TIP core	–	o	+	–	o
TIP 2.9	+	+	+	+	+

Table 1: Comparison of selected traveller information systems. (Symbols: + supported feature, – not supported feature, o supported in part, ? no information)

of modular incorporation of and cooperation between various (existing) services. TIP satisfies existing and new requirements regarding modelling, architecture, and application domain. The proposed architecture can be realized in a number of ways; we tried to built on standard technologies as far as possible. We evaluated several implementation variations and reported our lessons learned and discussed the issues requiring further research.

Related Approaches from Other Areas For the coordination of web services, Alvarez at al. (Álvarez, Bañares & Muro-Medrano 2003) propose an extension of service-oriented architectures with a coordinator role that allows more flexible relationships between service providers and requestors than the one provided by the client-server model. This new role has a similar function to our client-broker/ENS middleware. A component for coordinating changes in web services has been proposed in (Hinze 2005); this is similar to the coordinating role of the ENS in our architecture.

Similar issues as the ones discussed here have been addressed in the area of agent-based systems (e.g., in (Zlotkin & Rosenschein 1989)). Cooperation and coordination between multiple agents is a fundamental question that has been dealt with in a wide variety of research, such as telecommunications (Magedanz, Rothermel & Krause 1996), for trading (Chavez & Maes 1996), and dynamic network configuration (Minar, Kramer & Maes 1999). The central focus lies on the theory of agent communication and negotiation, drawing from such areas as artificial intelligence and game theory. The difference to our approach is that cooperating agents are inherently designed for communication and cooperation, whereas the services in our application field are mainly designed to be executed on their own or to be cooperating with dedicated services. The cooperation and management has to be provided by auxiliary components in the architecture (similar to the management of web services).

General context-aware systems in ubiquitous computing have already been discussed in earlier works (Schilit, Adams & Want 1994). There, context refers to individuals that interact with computers, keyboards and mice. As a whole, they are seen as a reconfigurable system that is driven by context information. Complex cooperation of concurrent services has not been in the focus of this work. An overview about approaches to context aware system can be found in (Chen & Kotz 2000). Context is mainly the location of the user; more complex concepts such as the user interests are rarely considered.

7 Conclusion

In this paper, we have introduced a mobile infrastructure for cooperating information services, demonstrated through the example of a mobile Tourist Information Provider (TIP) system. TIP delivers context-sensitive information to the system's users. The infrastructure uses an modular approach combined with an event-based communication layer to support continually changing information.

We identified a number of the challenges for creating the infrastructure for mobile systems, namely the support of modular, cooperating, mobile services for the information delivery to mobile users. We presented the conceptual design and implementation details of our mobile tourist information service TIP 2.9.

We presented details of two service types (map service and recommender services) that required different client implementations. We highlighted the challenges of supporting services that require thin and thick clients on a single device. These services need to communicate with each other on the mobile device and with the central TIP server. This paper reported on our exploration of various process management structures to achieve flexible cooperation between these services. We illustrated how our modular, service-oriented approach allows for the exploration of alternatives in process communication and implementation. The conceptual architecture given in Section 3.1 has proven effective in delivering the requirements we introduced at the beginning of this paper, providing a framework for both modularity and cooperation.

This paper presented our TIP 2.9 prototype of a mobile tourist information system that implements our design of a mobile infrastructure for cooperating information services. In our comparison of our TIP 2.9 system to related existing systems, we have shown that none of the other systems fully address the problems of modular incorporation of and cooperation between various (existing) services in a mobile information delivery system. This is exemplified in the case of recommender systems; our implementation of recommendation support is able to provide a much wider set of options than any existing system, and does so through exploiting our inter-service communication capacity. When a particular recommendation approach cannot function – e.g., through the lack of availability of appropriate data – its peers can continue to provide the same service type through a different implementation.

In future work, we wish to extend the co-operation (and thus communication) between the provided services. We also plan to incorporate new services, such as access to external information sources, e.g., in digital libraries. This may lead to further exploration of sophisticated context-models which can be used for standardized communication between the services. We wish to support even more flexible service utilization: services may register and unregister depending on availability and capability of the mobile device. The next step in the TIP design will therefore be the completion of the re-design into a Service-Oriented Architecture (SOA) using web services in TIP 3.0.

Acknowledgements This work was valuably assisted by our colleagues Dr. Matt Jones, Dr. Steve Jones and Bruce Bowering who created the original Zoomable Map Interface. The Royal Society of New Zealand supported this work through an ISAT grant award.

References

Abowd, G., Atkeson, C., Hong, J., Long, S., Kooper, R. & Pinkerton, M. (1997), 'Cyberguide: A mobile context-aware tour guide', *ACM Wireless Networks* **3**, 421–433.

Álvarez, P., Bañares, J. A. & Muro-Medrano, P. R. (2003), An architectural pattern to extend the interaction model between web-services: The location-based service context., *in* 'First International Conference on Service-Oriented Computing', Trento, Italy.

Bainbridge, D. & Witten, I. H. (2004), Greenstone digital library software: current research, *in* 'JCDL '04: Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital libraries', Tuscon, AZ, USA.

Buchanan, G. & Jones, M. (2000), Search interfaces for handheld web browsers, *in* 'Poster Proceedings of the 9th World Wide Web Conference', Amsterdam, Netherlands.

Chavez, A. & Maes, P. (1996), Kasbah: An agent marketplace for buying and selling goods, *in* 'First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'96)', London, UK.

Chen, G. & Kotz, D. (2000), A survey of context-aware mobile computing research, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.

Cheverst, K., Mitchell, K. & Davies, N. (2002), 'The role of adaptive hypermedia in a context-aware tourist guide', *Communications of the ACM* **45**(5), 47–51.

Hinze, A. (2005), Supporting change-aware semantic web services., *in* 'Proceedings of the First Workshop on Service Oriented Computing', Leicester, U.K.

Hinze, A. & Buchanan, G. (2005), Cooperating Services in a Mobile Tourist Information System, *in* 'Proceedings of the Conference on Cooperative Information Systems (CoopIS)', Agia Napa, Cyprus.

Hinze, A. & Junmanee, S. (2005), Providing recommendations in a mobile tourist information system, *in* 'Information Systems Technology and its Applications, 4th International Conference (ISTA 2005)', Palmerston North, New Zealand.

Hinze, A., Loeffler, K. & Voisard, A. (2004), Contrasting object-relational and RDF modelling in a tourist information system, *in* 'Proceedings of the 10th Australian World Wide Web Conference', Gold Coast, Australia.

Hinze, A. & Voisard, A. (2003), Location- and time-based information delivery in tourism, *in* 'Conference in Advances in Spatial and Temporal Databases (SSTD 2003)', Vol. 2750 of *LNCS*, Santorini Island, Greece.

Jones, S., Jones, M., Marsden, G., Patel, D. & Cockburn, A. (2005), 'An evaluation of integrated zooming and scrolling on small-screens', *International J. Human Computer Studies* . In Press.

Jung, D. & Hinze, A. (2005), A mobile alerting system for the support of patients with chronic conditions, *in* 'Proceedings of the 1st Euro Conference on Mobile Government mGov'2005', Brighton, UK.

Klante, P., Krsche, J. & Boll, S. (2004), AccesSights – a multimodal location-aware mobile tourist information system, *in* 'Proceedings of the 9th International Conference on Computers Helping People with Special Needs (ICCHP'2004)', Paris, France.

Magedanz, T., Rothermel, K. & Krause, S. (1996), Intelligent agents: An emerging technology for next generation telecommunications?, *in* 'INFOCOM'96', San Francisco, CA, USA.

Minar, N., Kramer, K. H. & Maes, P. (1999), Cooperating mobile agents for mapping networks, *in* 'Proceedings of the First Hungarian National Conference on Agent Based Computation'.

O'Hare, G. & O'Grady, M. (2003), 'Gulliver's genie: A multi-agent system for ubiquitous and intelligent content delivery', *Computer Communications* **26**(11), 1177–1187.

Ottlinger, P. (2004), Design and Implementation of an extensible Software architecture for Distributing context-sensitive Information (in German), Master's thesis, Freie Universitaet Berlin, Department of Computer Science.

Pashtan, A., Blattler, R. & Heusser, A. (2003), Catis: A context-aware tourist information system, *in* 'Proceedings of the 4th International Workshop of Mobile Computing', Rostock, Germany.

Poslad, S., Laamanen, H., Malaka, R., Nick, A., Buckle, P. & Zipf, A. (2001), CRUMPET: Creation of user-friendly mobile services personalised for tourism, *in* 'Proc. 3G2001 Mobile Communication Technologies', London, U.K.

Schilit, B., Adams, N. & Want, R. (1994), Context-aware computing applications, *in* 'IEEE Workshop on Mobile Computing Systems and Applications', Santa Cruz, CA, USA.

Warren, N., Jones, M., Jones, S. & Bainbridge, D. (2005), Navigation via continuously adapted music, *in* 'CHI '05 extended abstracts on Human factors in computing systems', Portland, OR, USA.

Zlotkin, G. & Rosenschein, J. S. (1989), Negotiation and task sharing among autonomous agents in cooperative domains, *in* N. S. Sridharan, ed., 'Proceedings of the Eleventh International Joint Conference on Artificial Intelligence', Morgan Kaufmann, San Mateo, CA, pp. 912–917.

Human Visual Perception of Region Warping Distortions

Yang-Wai Chow[†], **Ronald Pose**[†], **Matthew Regan**[†], **James Phillips**[††]
School of Computer Science and Software Engineering[†] / Department of Psychology[††]
Monash University
Clayton, Victoria 3800, Australia
{yang.wai.chow,rdp,regan}@csse.monash.edu.au, jim.phillips@med.monash.edu.au

Abstract

Interactive virtual reality requires at least 60 frames per second in order to ensure smooth motion. For a good immersive experience, it is also necessary to have low end-to-end latency so that user interaction does not suffer from perceptible delays in images presented to the eyes. The Address Recalculation Pipeline (ARP) architecture reduces end-to-end latency in immersive Head Mounted Display (HMD) virtual reality systems. By using the ARP in conjunction with priority rendering, different sections of the scene are updated at different rates. This reduces the overall rendering load and allows for more complex and realistic scenes. Large object segmentation in conjunction with priority rendering further reduces the overall rendering load. However, scene tearing artefacts potentially emerge and region warping was devised to alleviate this. In compensating for the tearing, region warping introduces slight distortions to the scene.

Immersive virtual reality systems have humans as integral parts of the system. While researchers do thorough measurements and evaluation of hardware and software performance, the human experience and perception of the system is often neglected. This paper addresses this important issue. We describe our human visual perceptual experimental methodology in detail and present some initial results. Initial experiments in human visual perception of region warping distortions show interesting characteristics which lead us to propose further experimental investigations to clarify their significance.

Keywords: Address Recalculation Pipeline, object segmentation, priority rendering, region warping, tearing artefacts, visual perception.

1 Introduction

The ultimate goal of virtual reality is to present the user with an illusion of reality within a virtual environment. One of the main aspects involved in the portrayal of a believable virtual environment is the presentation of computer generated 3D imagery to the user. In order to maintain the illusion of reality, virtual reality systems must continually display images from the user's vantage point in real time. Interactive virtual reality requires at least 60 updates per second to achieve good immersion.

Latency is a well recognized problem in virtual reality and teleoperation technology (Ellis et al. 2004). Latency is the lag or time delay between when a user performs an action and when the system responds to that action or when that action is represented by the system. A particular issue in virtual reality is that of the latency between a user moving his/her head, thus changing the user's viewpoint, and that change being reflected in updated images before the user's eyes. Excessive system latency or delays in virtual reality makes the system hard to use and in severe cases this can lead to adverse side effects such as user disorientation, motion sickness, and etc. Even with the fast graphics accelerators available today that can render over 100 frames per second (fps), latency still remains a factor that has to be addressed (Meehan et al. 2003). This is because on conventional systems the update cycle is bound by the need to obtain user head position orientation information before rendering can commence.

A hardware architecture known as the Address Recalculation Pipeline (ARP) was designed to reduce user head rotational latency in immersive Head Mounted Display (HMD) virtual reality by detaching user head orientation from the rendering process (Regan and Pose 1993). Priority rendering was developed for use in conjunction with the ARP system in order to reduce the overall rendering load by concentrating rendering power on sections of the scene that appear to change the most. Using priority rendering, different sections of the scene can be updated at different rates. This allows for the rendering of more complex and realistic scenes (Regan and Pose 1994).

Large object segmentation and region priority rendering were introduced to manage objects in the virtual environment as well as to further reduce the overall rendering load (Chow et al. 2005a). However, tearing problems can emerge as a result of updating different segments of the same object at different rates. This problem was overcome by using region warping. Region warping involves the perturbation of object vertices in model space in order to force these vertices to align, thereby hiding scene tearing artefacts. This however introduces slight distortions in the computer generated graphics.

The quality of frames produced by implementing the region warping technique has previously been investigated using mathematical analysis (Chow et al. 2005b). However while mathematical analysis gave an indication of the level of distortion caused by region warping, it did not incorporate the human visual perception of the distortions. It is therefore essential to conduct perception experiments in order to fully understand the characteristics of region warping distortions with respect to human visual perception. Virtual reality and interactive systems involve humans as integral parts of the systems. All too often researchers thoroughly test and measure the performance of the hardware and software while ignoring the fundamental issue of how humans might perceive and react to the system. This paper addresses this issue by presenting the experimental methodology and results of an experiment investigating the human perception of region warping distortions.

The results and insights gained in the research presented in this paper are also relevant in other areas and applications of computer graphics. There are a number of other perceptually based computer graphics techniques that attempt to optimize a system's performance taking advantage of the limitations in the human visual system. Some of these applications are provided in the related work section of this paper.

2 Previous Work

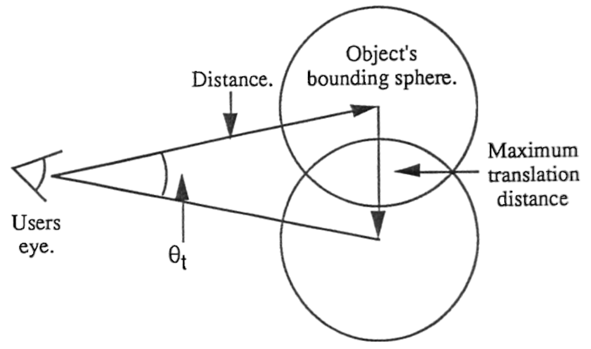
In order to describe the basis for region warping, this section first provides some background of priority rendering and the Address Recalculation Pipeline system. For more information please refer to Pose and Regan (1994), Regan and Pose (1993, 1994), Chow et al. (2005a).

2.1 The Address Recalculation Pipeline and Priority Rendering

The Address Recalculation Pipeline (ARP) graphics display architecture reduces the end-to-end latency perceived by the user during user head rotations, by implementing a concept known as delayed viewport mapping. In delayed viewport mapping, the scene that encapsulates the user's head is pre-rendered into display memory. Viewport orientation mapping is then performed only when required by mapping relevant sections of the scene already residing in display memory, and is therefore fairly independent of scene complexity and is based on the most up-to-date user head orientation information. Unlike conventional systems where user head orientation has to be obtained prior to rendering, the ARP system effectively detaches user head orientation from the rendering process. Thus in the ARP system, latency is no longer bound by the usually lengthy rendering process.

A rendering technique known as priority rendering was designed to be used in conjunction with the ARP system. Priority rendering takes advantage of the fact that in the ARP system, the scene surrounding the user's head has to be rendered into display memory. In a scene that surrounds the user's head, only dynamically animated objects might constantly be changing, when the user rotates his/her head. Most other objects in the scene will remain the same. When a user translates through a scene, sections of the scene that are closer to the user will appear to change faster than sections that are further away from the user. This difference in the speed of movement for near and far objects is known as motion parallax (Goldstein 1999). Therefore by using multiple display memories and multiple renderers, different sections of the scene can be rendered onto separate display memories that can be updated individually at different rates. The images on the different display memories can then be composited to form an image of the whole scene.

Priority rendering is demand driven rendering, in that an object is not updated until its image in display memory has changed above a tolerable amount. This concentrates rendering power on sections of the scene that are changing the most. In this manner, priority rendering reduces the overall rendering load, thus potentially allowing for the rendering of more complex and realistic scenes. A threshold which defines the minimum feature size of the virtual environment has to be pre-determined. This threshold takes the form of an angle, θ_t. Priority rendering attempts to keep the image in display memory accurate to within this threshold. Figure 1 shows a priority rendering translational validity period estimation. The translational validity period is an estimate of how long the image of an object in display memory will remain valid before requiring an update, based on the user's relative translational speed. The translational validity period estimation was used in the experiment by varying the value of θ_t for the different scenes.



$x^2 = d^2 + d^2 - 2d*d*\cos(\theta_t)$

$x^2 = 2d^2(1-\cos(\theta_t))$

$x = d*\text{sqrt}(2(1-\cos(\theta_t)))$ where d = distance, and x = max. trans. distance

translational validity period = x / relative_speed

Figure 1: Translational validity period estimation.

2.2 Large Object Segmentation and Region Priority Rendering

Region priority rendering was introduced to assist object management in the virtual environment for priority rendering. In region priority rendering, objects in the virtual world are spatially divided and grouped into different regions. Using this technique, entire regions of objects can then be assigned to the separate display memories based on the region that they are located in. This avoids having to calculate individual object validity periods and sort objects based on these validity periods before assigning individual objects to the separate display memories. Figure 2 shows a top-down example of region to display memory allocations for region priority rendering. This allocation strategy was employed for the virtual environment used in the experiment.

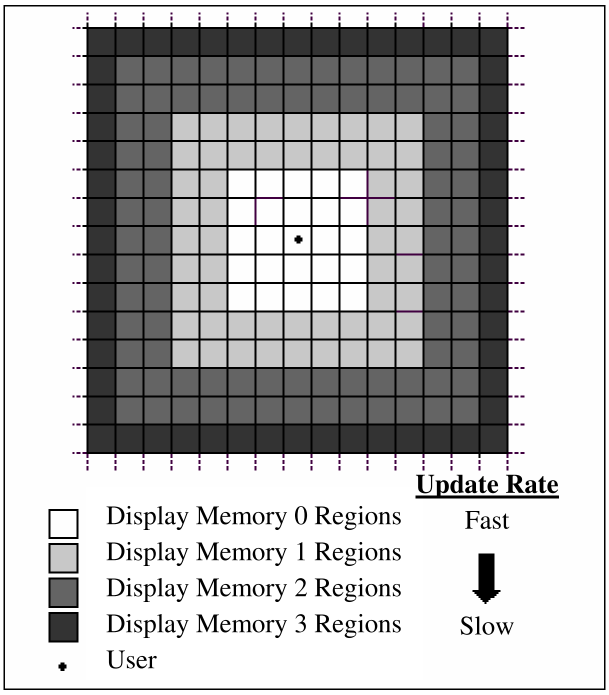


Figure 2: 2D top-down view of example region to display memory allocations.

Large object segmentation was devised to further reduce the overall rendering load in priority rendering. Large virtual world objects were segmented so that different segments of these objects could be updated on separate display memories at different update rates. In this manner, if the image of a section of a large object became invalid in display memory and required an update, only this section of the object would have to be updated instead of having to re-render the whole object.

2.3 Tearing and Region Warping

The implementation of large object segmentation with region priority rendering however causes an adverse visual artefact, a form of scene tearing. Scene tearing artefacts occur along the shared vertices of objects' segments. User head rotations will not cause the scene surrounding the user's head to appear to change much. The problem occurs when the user translates through the scene. Tearing artefacts emerge as a result of discrepancies between the shared vertices of an object's segments that are updated on separate display memories at different update rates, whilst the user is translating through the scene. This tearing problem can destroy the illusion of reality that the virtual reality system attempts to present to the user. An example of scene tearing artefacts is shown in figure 3 (the white lines in the scene).

Region warping was devised in order to alleviate the tearing problem. Region warping involves the perturbation of shared object vertices in order to force these vertices to align, thereby eliminating any discrepancy between the vertices during user translations. Two methods of interpolation for region warping were introduced in Chow et al. (2005b), linear interpolation and quadratic interpolation. Before warping could be performed, the exact amount of perturbation had to be known. Vertices in the regions had to be normalized based on their distance from the user's region. This is illustrated in figure 4.

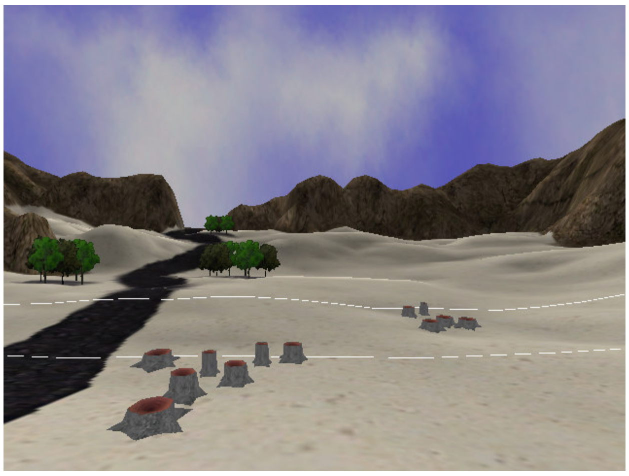


Figure 3: Example of scene tearing artefacts.

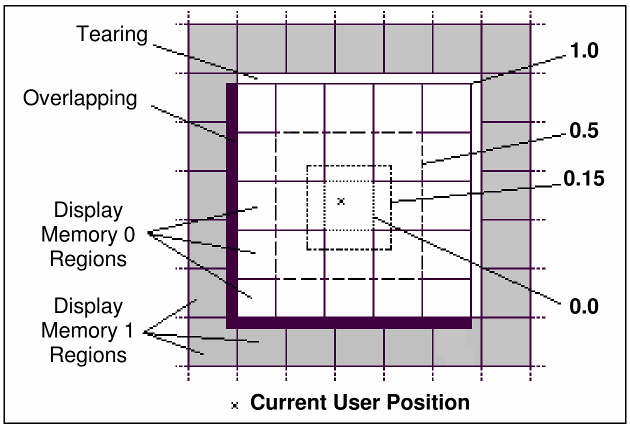


Figure 4: 2D top-down view illustrating the normalization of region vertices.

Normalization was performed using what can be seen as concentric squares, centered on the circumference of the square based region the user was located in. Interpolation of vertex perturbations could then be conducted with these normalized values. In linearly interpolated region warping, these normalized values were used without any alteration, whereas in quadratic interpolation the square of the normalized values was used. Both linear and quadratic region warping methods were tested in this experiment. In Chow et al. (2005b) it was concluded that the distortions in the frames generated using quadratic region warping were concentrated further away from the user and were more similar to normally rendered frames, compared to frames rendered using linear region warping. This conclusion however was based on mathematical comparisons and did not reflect human visual perception of the distortions.

3 Related Work

Similar concepts of using multiple renderers and/or multiple display memories have also been designed and developed by other researchers.

NVIDIA Corporation's Scalable Link Interface (SLI) technology combines the rendering power of two Graphics Processing Units (GPUs) in a single system (NVIDIA 2005). The SLI system has a rendering mode called Split Frame Rendering (SFR) which allows a frame to be divided into two portions (top and bottom) and rendered separately on each GPU. Software drivers are used to dynamically share and balance the load between the two GPUs. Each GPU then renders one of the two sections, before the sections are digitally composited to form the whole frame. By clipping the scene into 2 portions, the system attempts to avoid the processing of all the vertices in a frame on both GPUs.

A 3D graphics and multimedia hardware architecture codenamed Talisman was designed by researchers at Microsoft Corporation (Toburg and Kajiya 1996). One of the main uses of the Talisman architecture was in multimedia applications such as interactive animation. In smooth animated sequences, most of the display remains the same from frame-to-frame and it would be wasteful to have to re-render the entire frame. The Talisman architecture takes advantage of temporal and spatial similarities between sequential frames, by allowing individually animated objects to be rendered onto independent image layers before being composited together to form the final display (Barkans 1997). In this way, only changing image layers have to be modified or re-rendered.

3D warping techniques in computer graphics have also been looked into by a number of other researchers. Mark et al. (1997) have experimented on what they termed, post-rendering 3D warping, on adjacent frames in order to avoid re-rendering entire frames by exploiting frame-to-frame coherence. The purpose of this warping was to increase the frame rate of a graphics system. This was done by warping the images of existing frames in order to extrapolate for new viewpoints of future frames. Thus, less rendering had to be performed for the derived frames resulting in an increase in the rendering frame rate. They have also suggested that the priority rendering technique for the ARP system, and also the Talisman architecture, would benefit from the implementation of 3D warping techniques.

In light of the fact that the human visual system can only perceive a limited amount of detail, perceptually orientated graphics techniques have been designed to optimize a system's performance. For example, Level of Detail (LOD) management techniques attempt to remove or reduce less perceptible details from the computer graphics (Reddy 1997). Watson et al. (1997) have experimented on the effects of degrading the peripheral detail of a scene in a HMD virtual reality system's display with respect to user performance. Visual perception experiments using visual attention models have also been conducted to predict where a user will look in a scene and to selectively concentrate computational effort on those sections of the scene (Chalmers et al. 2003).

4 Psychophysics Experiment

The aim of this experiment was to measure the threshold where a human can perceive the distortions caused by region warping. Other goals of the experiment were to determine whether different display devices and/or region warping methods might affect the human perception of these distortions. Psychophysical methods of testing were therefore employed for the experiment. Psychophysics is a branch of psychology that deals with the measurement of perception. It is the scientific study of the relation between stimulus and sensation (Gescheider 1985).

4.1 Method

The method used for the experiment was a variation of the method of limits known as an adaptive staircase Two Alternative Forced Choice (2AFC) method. An adaptive procedure means that the stimulus of a trial is determined by the preceding stimuli and responses (Levitt 1971). The advantage of using this procedure is that trials will be concentrated around the area of interest, i.e. near the threshold, and therefore presents an efficient method for measuring threshold.

The staircase 2AFC technique has been widely used in a variety of different fields to measure threshold of perception. Experiments using similar methodologies have also been conducted in the field of virtual reality and computer graphics by other researchers, for example, to measure perception of latency in virtual reality (Regan et al. 1999, Ellis et al. 2004, Mania et al. 2004), and in perception of modulated Level-of-Detail (LOD) in computer graphics (Reddy 1997).

In a 2AFC discrimination task, two stimuli are presented to the participant - a standard (S) and a comparison (C) (Ulrich and Miller 2004). The participant is then required to choose which of these two alternatives, contained (or did not contain) the signal. The comparison stimulus presented for each trial varies in signal strength, and the number of different strength values is usually 5 or 7 (Farell and Pelli 1999).

The forced-choice method was chosen for the experiment over yes/no or same/different methods, as these other methods contain an internal subjective criterion (Farell and Pelli 1999). In a yes/no or same/different experiment, for each trial the participant would be asked whether they could discern a difference between the two stimuli or whether the stimuli were the same or different. This might introduce a certain bias to the experiment, for example, the participant's responses might be influenced if he/she knew the purpose of the experiment beforehand. The 2AFC method attempts to eliminate this bias, by forcing the participant to choose between the two alternatives.

The staircase or up-down method used for the experiment followed a 2-Down 1-Up (2D-1U) approach, where two correct responses reduced the signal strength whereas an incorrect response increased or augmented the signal strength. This method gives a 70.7% correct response threshold (Levitt 1971). Descending and ascending staircases were used in the experiment. A descending staircase is one which starts at a high/strong signal strength, and the signal strength is typically reduced depending on the participant's responses. An ascending staircase starts at a low/weak signal strength, and the signal strength is typically increased based on the participant's responses (Gescheider 1985). An example of ascending and descending staircases can be seen in figure 7, in the results and discussions section below.

4.2 Design

For this experiment, the standard stimulus was the scene rendered using normal rendering while the comparison stimulus was the scene rendered using the region warping technique, and containing varying levels of distortion. For each trial, the two scenes, scene A and scene B, were presented sequentially one after the other. This is known as temporal-forced choice (Gescheider 1985). The order of the two scenes (i.e. the standard and comparison stimulus) presented to the participants was pair-wise randomized. Participants were then forced to choose which of the scenes appeared better, in other words, which scene did not contain distortions (the standard stimulus). Participants used a 2 button mouse to input their forced-choice feedback.

There were 2 parts to the experiment. Each part required the participant to carry out the experimental tasks on a different display device. The two display devices used were: a 17 inch computer monitor and a Head Mounted Display (HMD). The HMD used for the experiments was a Virtual Research V8 HMD, which has a 640×480 resolution and a 60Hz refresh cycle. The other display device was a 17 inch computer monitor set to a resolution of 640×480 and a 60Hz refresh cycle, in order to match the HMD's resolution and refresh rate. In view of the fact that the other device used was a computer monitor, the HMD was used in monoscopic mode. The wide angle lens optical distortion of the V8 HMD was corrected using a technique proposed by Watson and Hodges (1995).

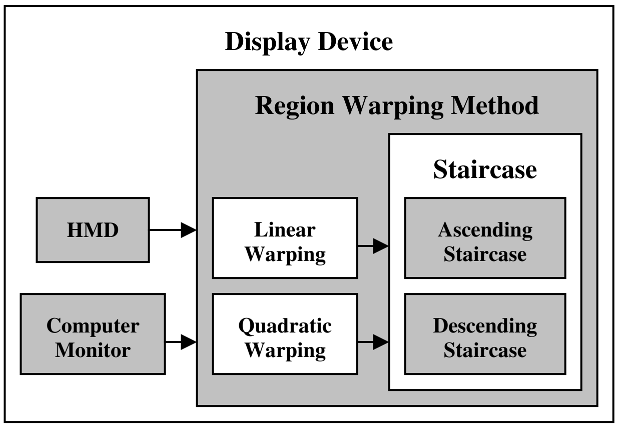


Figure 5: Overview of experimental design.

A pair of staircases (one ascending, one descending) was used for each region warping technique (i.e. linear and quadratic), giving a total of four staircases. Both ascending and descending staircases were used in this experiment in order to comprehensively test the participants' responses to the full range of distortion levels. All four staircases were randomly interleaved based on a pre-randomized script. This was done so that participants would not know which warping method was being presented during each trial and also to minimize the chances of the participant anticipating or predicting the signal strength of the next trial. For example, if the participant realized that he/she was on an ascending or descending staircase he/she would be able to anticipate whether successive trials would increase or reduce in signal strength. Figure 5 depicts the overall experimental design. The entire experiment was automated on the computer.

Fixed-step staircases with 7 signal strengths (distortion levels) were used. The signal strengths corresponded to the level of region warping distortions. Each staircase would end after 6 reversals or a maximum of 25 trials, whichever came first. A reversal refers to a change in the direction of a staircase. In a 2D-1U staircase approach, a reversal would mean 2 correct responses after 1 (or more) incorrect responses or an incorrect response after 2 (or more) correct responses. An example of this is shown in figure 7, where the ascending staircase ended after 6 reversals while the descending staircase ended after 25 trials. This meant that each part of the experiment would finish after a maximum of 100 trials. It is important to note that unlike experimental simulations that can be done purely on a computer, in experiments involving humans one must design the experiment to avoid exhausting the participants.

The V8 HMD has a 60^0 diagonal Field of View (FOV), which gives a 48^0 horizontal FOV and a 36^0 vertical FOV. Since the HMD has a 640×480 resolution, this equates to 4.5 arc minutes per pixel (Mania et al. 2004). This means that a θ_t value of 4.5′ or 0.075^0 corresponds to 1 pixel. The 7 levels of distortion used in the experiment were obtained by varying θ_t (refer to θ_t in the translational validity period equation shown in figure 1) in step sizes of ½ a pixel. Table 1 shows the relations between distortion levels, θ_t, and the number of pixels.

Distortion Level	Value of θ_t (arcmin)	Pixels
1	2.25′	½
2	4.5′	1
3	6.75′	1½
4	9.0′	2
5	11.25′	2½
6	13.5′	3
7	15.75′	3½

Table 1: Value of θ_t for each signal strength.

By using these θ_t values, a display memory would be forced to update if the image in that display memory was no longer valid to within θ_t from the user's viewpoint. A view of the virtual environment used in the experiment can be seen in figure 11. Region sizes in the virtual environment were 10 square metres. The scenes were rendered using a simulation of region priority rendering.

Each scene in the experiment had the camera or viewport translating along a fixed path. This was to ensure that all participants would view the exact same sequence of frames. The worst case scenario was chosen for the fixed path. In other words, the path chosen would give the maximum distortions with the viewport looking in the direction where the distortions would be most apparent. The fixed path included both horizontal and vertical movements, with the viewport translating through the scene at a constant velocity of 1.5 meters per second (a fast walking speed). Figure 6 illustrates the update rates used for a θ_t value of 4.5′ at this translational speed (refer to figure 2 for the relative locations of display memory 0, 1, 2 and 3 regions with respect to the user). The display memories had different update rates for other values of θ_t. Nevertheless, display memory 0 was always kept at 60 updates per second. While the fixed path movement used in the experiment may affect user visual cues, such as depth perception, user visual perception of region warping distortions based on user initiated movement is the subject of further study.

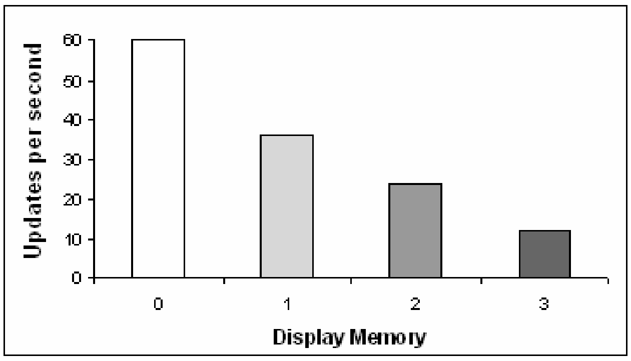


Figure 6: Display memory update rates for the scene using a θ_t value of 4.5′.

Each scene was 3 seconds in duration. The scenes had to be short as two scenes were presented sequentially to the participants per trial. This meant that the participant had to remember what he/she saw of the two scenes before deciding which of the scenes appeared to look better. This was also done in the interest of keeping the experiment short, so that the participants would not lose concentration during the experiment.

No anti-aliasing techniques were used in the rendering of the virtual environment. Anti-aliasing is used in computer graphics in order to smoothen out aliasing artefacts such as jagged edges. Anti-aliasing also has the effect of smoothening out the pixel transitions of moving objects. For this reason, it is expected that region warping distortions will be less obvious if anti-aliasing were to be used. However, this was left for future experiments. Note that hardware anti-aliasing is part of the design of the ARP system.

4.3 Procedure

A total of 16 volunteers (12-male and 4-female) participated in the experiment. Participants were aged between 18-41, and had normal vision or corrected normal vision. All participants were experienced computer users.

Participants were informed as to the purpose of the experiment, which was to determine whether or not they could tell the difference between the different methods of rendering. They were told that even though the same virtual environment was used for all the trials, the two scenes presented in each trial were different, and that for each trial they were to choose which of the scenes looked better. Participants were instructed to guess if they could not tell which of the two scenes was better.

Before the experiment, participants were shown an example of region warping distortions during a test run of the experiment. A different virtual environment was used for the example, so that it would not influence the actual experiment. The purpose of the test run was to familiarize the participants with the experiment, what they were required to do, as well as what they were expected to look for during the experiment. However, even though participants were shown region warping distortions on the virtual environment used for the test run, participants were not instructed as to what criterion to adopt when looking for distortions in the virtual environment employed for the actual experiment and therefore had to judge for themselves which of the two scenes contained the distortions.

Participants were encouraged to take a rest break in between the two parts of the experiment, i.e. when switching from one display device to the next. This was to ensure that the participants would remain attentive throughout the experiment. Some participants also took rest breaks at various occasions in between trials. The duration of the whole experiment was approximately 30-40 minutes per participant. In order to balance the experiment, half the participants used the HMD first then the computer monitor, while the other half used the computer monitor first followed by the HMD. This was done to balance any ‘learning effects’, i.e. participants would more likely perform better during the second part of the experiment as he/she would be more familiar with the experimental tasks and would know where to focus their attention in the virtual environment.

After the experiment, participants were required to provide some personal computer usage information as well as to answer some questions relating to the experiment by filling in a questionnaire.

5 Results and Discussions

Of the experimental data collected from the 16 participants, only 15 sets were used in the analysis. The experimental results of one of the participants who consistently answered incorrectly even at the highest distortion level were not considered in the analysis. Prior to this experiment, only one of the participants had ever used a HMD.

In the experimental results, a *higher* threshold meant that the participant was *less* likely to be able to correctly perceive a difference in the scenes presented during each trial of the experiment, whereas a *lower* threshold meant that the participant was *more* likely to be able to correctly differentiate between the scenes. From observations of the individual participant responses, it was clear that region warping distortions of level 5, corresponding to 2½ pixel distortions, and above were generally perceptible by the participants.

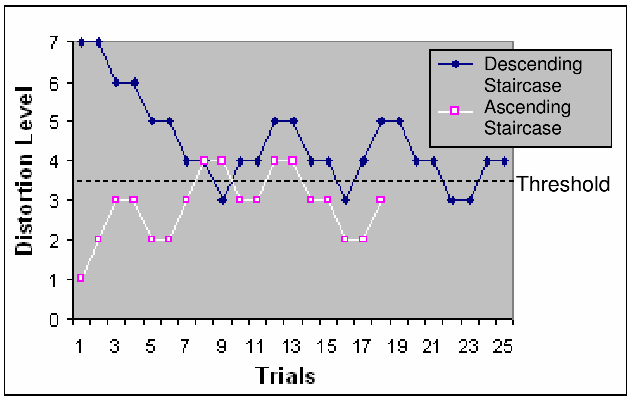


Figure 7: A plot showing the ascending and descending staircases for one of the participants.

From the experimental data, graphs of the ascending and descending staircases were plotted for the different display devices and the different region warping methods. This gives a total of 4 graphs for each individual participant. Figure 7 shows an example of this. The mid-run estimation method was used to determine the threshold for the individual participants. Mid-run estimations are the average of the mid-points of staircase runs. A staircase run is an ascending or descending sequence in the staircase. This means that a staircase reversal would mark the end of a staircase run and the start of another run. The mid-run method of threshold estimation is an efficient way of estimating threshold and the precision of the mid-run estimates has been found to be excellent for small experiments of less than 30 trials (Levitt 1971). In the analysis, the first staircase run for each staircase was ignored in order to give the staircases a chance to stabilize.

Display Device	Warping Method	Mean	Standard Deviation
Monitor	Linear	2.7409	0.84552
	Quadratic	2.6910	1.18052
HMD	Linear	3.1789	1.07441
	Quadratic	3.2867	0.75123

Table 2: Means and standard deviations of participants' thresholds.

Table 2 shows the means and standard deviations of the participants' thresholds for both display devices and both methods of region warping. It can be seen from the table that there is not much difference in average threshold for the different warping methods for each display device respectively. However there is a clear difference in average threshold between the different display devices. The HMD has a higher mean threshold than the computer monitor, which indicates that participants found distortions harder to see on the HMD as compared to the monitor.

Upon closer examination of the thresholds, it appeared that in general participants who indicated on the questionnaire that they frequently played computer games (this includes console gaming platforms like the X-Box, Play Station, and etc.) had a lower threshold compared to participants who answered to the contrary. A total of 6 participants indicated that they were gamers. Results of a t-test only reported a statistically significant difference in threshold for the HMD and linear warping case [$t(13) = 4.93$, $p<0.01$], the box plot in figure 8 depicts this difference. Nevertheless, differences in average threshold levels between gamer and non-gamers can clearly be seen in figure 9. This result is not surprising as gamers are more attuned to interactive graphics.

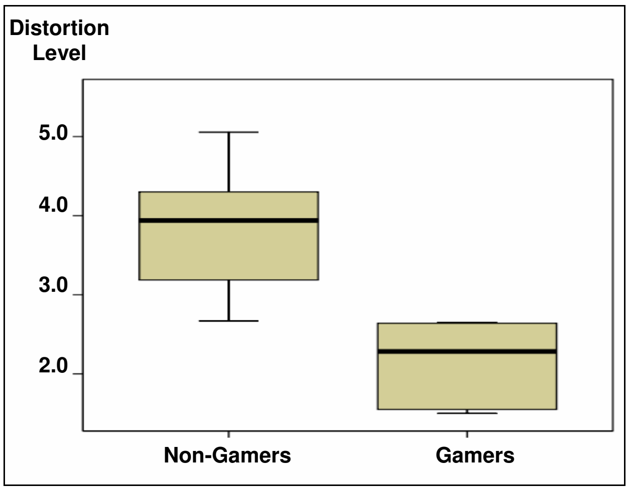


Figure 8: Box plot depicting the difference in threshold between gamers and non-gamers for the case of linear warping viewed using the HMD.

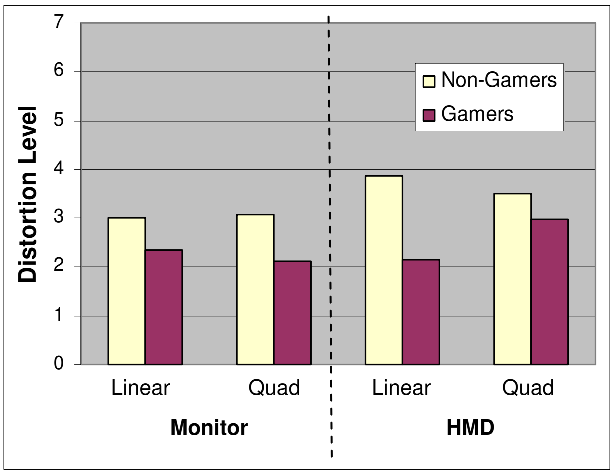


Figure 9: Graph comparing mean threshold between gamers and non-gamers for different display devices and different warping techniques.

In order to further investigate the threshold differences between gamers and non-gamers, group thresholds for the gamers and non-gamers were calculated. Note that mid-point estimations could not be performed for group thresholds, as the staircases could only be plotted for individual participants. Therefore group thresholds were obtained by first calculating the probability of correct responses at each distortion level for the group, and then fitting a cumulative normal ogive to the groups' experimental results. As the cumulative normal ogive is an exponential function, a log transform was first applied to the average probability of correct responses for the two groups (to 2 minus the average probability of correct responses). A linear regression was then performed to obtain a fitted psychophysical function. This was performed for all cases involving the different warping methods and the different display devices.

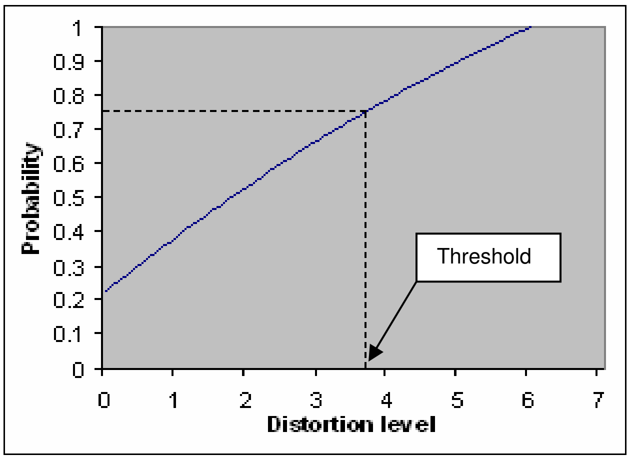


Figure 10: Group thresholds obtained from the 0.75 probability of the psychophysical function.

From fitted psychophysical functions the 0.75 probability was used to obtain the group threshold. For the 2AFC method, a probability of 0.5 signifies random guessing while a probability of 1.0 would mean 100% correct responses. 0.75 is typically used to obtain the threshold in 2AFC experiments (Ulrich and Miller 2004). Figure 10 illustrates the 75% threshold obtained from a fitted psychophysical function. Table 3 gives the results of the group thresholds.

	Monitor		**HMD**	
Gamer	***Linear***	***Quad.***	***Linear***	***Quad.***
	3.0558	2.3341	2.4372	3.6815
	Monitor		**HMD**	
Non-Gamer	***Linear***	***Quad.***	***Linear***	***Quad.***
	3.6886	3.8115	4.4975	4.3570

Table 3: Group thresholds between the gamers and non-gamers for the different cases.

From the table it can be seen that there is a clear difference in threshold between the gamers and non-gamers. This agrees with observations of the average individual thresholds between gamers and non-gamers presented earlier. For the non-gamers, the difference in threshold between the display devices also indicates that distortions on the HMD are not as perceptible as on the computer monitor, and that there is no apparent difference between warping methods. This also agrees with analysis presented earlier. However for the participants who were gamers, this was not the case. The results suggest that the gamers perceived less distortion in linear region warping on the computer monitor. But on the other hand, the gamers perceived less distortion in quadratic region warping on the HMD.

At this stage it is not clear as to why the difference in warping technique on the different display devices for the gamers is so apparent. Green and Bavelier (2003) have also found that habitual gamers have a perceptually modified visual attention compared to non-gamers. One possible explanation might be due to the size of the display area of the display device. Plumert et al. (2004) have observed that user perception of distance judgment in virtual environments is better on large screen immersive displays rather than virtual reality systems involving HMDs. Results of another study by Willemsen et al. (2004) that experimented with the use of HMDs and 'mock HMDs' [replica shell of a HMD that restricts a person's Field-of-View (FOV)], also implies that there is an apparent compression of perceived virtual spaces when using HMDs. Wu et al. (2004) have also previously conducted experiments with regards to human distance perception, by using a pair of opaque goggles with a monocular rectangular aperture to limit the horizontal or vertical FOVs of human subjects, and have reported similar findings.

These conclusions by other researchers also suggest that a user's sense of perception is affected by the use of HMDs. This is possibly why the gamers perceived warping distortions differently on the HMD as opposed to the monitor, and also why the non-gamers had higher thresholds on the HMD as compared with the computer monitor. Another reason might be because when set to a resolution of 640×480 in full screen, the area covered by 1 pixel will appear much larger on a 17 inch monitor compared to the HMD. In order to further investigate this, we are currently setting up visual perception experiments using two different HMDs with exactly the same resolution but with different FOVs, in order to affirm whether a HMD's FOV makes a difference to perception of region warping distortions.

In the questionnaire, participants were asked to circle a section or sections of the scene where they focused their attention the most. It was found that most participants focused their attention around one of two major sections in the virtual environment. Some participants circled both sections. These two sections are shown in figure 11 below and close ups of the two sections are provided in figures 12 and 13 respectively.

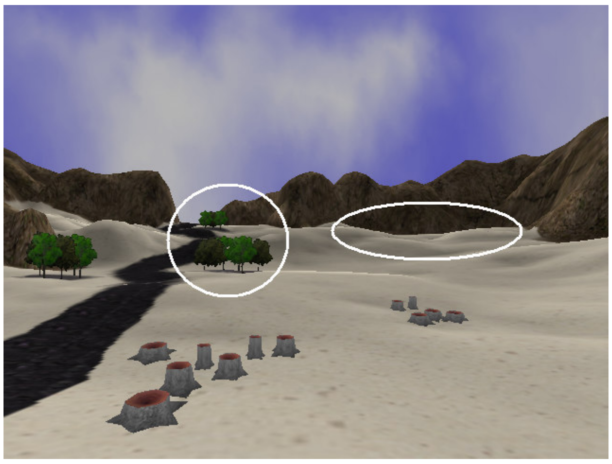


Figure 11: Sections of the virtual environment where participants focused their attention the most.

Chalmers et al. (2003) explains this focus of attention in virtual environments as being a fundamental feature of the human visual system known as inattention blindness. The center of the retina known as the fovea has the densest concentration of color sensitive cones in the human eye. The visual angle covered by the fovea however is very small, and everything that lies outside this foveal angle is perceived as blurred or unclear. Therefore in a virtual environment, a user will typically focus on conspicuous objects in the scene and will ignore or pay less attention to details in the rest of the virtual environment. This gives rise to the possibility of concentrating distortions away from where the user might focus his/her attention or to place eye catching objects in the scene in order to draw the user's attention away from the region warping distortions.

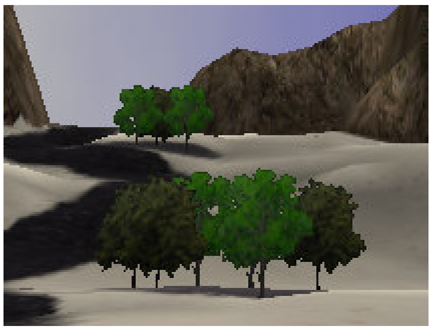


Figure 12: Close up of trees section.

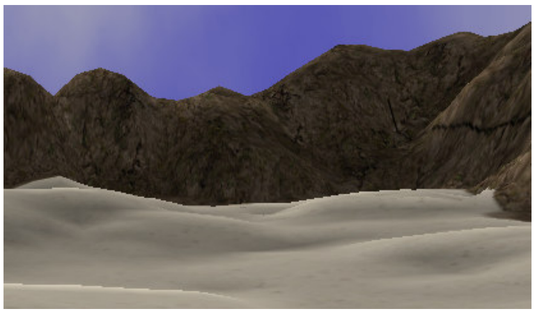


Figure 13: Close up of mountain-sand section.

Participants were also asked to choose whether the distortions were more noticeable during horizontal movement, vertical movement or whether direction of movement did not make a difference. It was interesting to note that participants who circled the trees section, typically indicated that distortions were more apparent during horizontal movement, whereas participants who circled the mountain-sand section typically selected vertical movement. When asked to clarify their answers, participants who circled the trees section said that they concentrated on the jerkiness in the movement of the trees during horizontal movement, and participants who circled the mountain-sand section said that they focused on the transition rate of the light and dark pixels between the mountain and the sand.

6 Conclusions and Future Work

The results of this experiment suggest that in general region warping distortions are less perceptible on a HMD compared to on a computer monitor. These findings concur with various other studies conducted by other researchers that conclude that the use of a HMD does affect a user's sense of perception. Distortions of over 2 pixels were generally perceptible by all the participants, whereas distortions below 2 pixels were less perceptible and distortions of 1 pixel and below were very much less perceptible.

Analysis of the experimental data also showed that there was a difference in visual perception between participants who frequently played computer/electronic games and participants who were non-gamers. It was also observed that a user will normally focus his/her attention on certain sections of a virtual environment and pay less attention to the remainder of the scene.

This work is relevant in a wide variety of fields in computer graphics research and applications, ranging from the design of interactive animation hardware like the Talisman to various perceptually based rendering techniques as well as other perception studies involving graphics and HMDs.

This experimentation is designed to enable us to concentrate our computing power on processes that improve the user experience. To that end we are now setting up visual perception experiments involving HMDs with different FOVs in order to further investigate human perception of region warping distortions. Considerations in planning for future experiments include employing anti-aliasing in the rendering of the virtual environment, as well as the use of different virtual environments with various scene characteristics and complexities. The impact of stereoscopy, different image resolutions and user-initiated interaction also has to be investigated. A discussion on a number of other factors that have to be considered when designing human visual perception experiments have previously been outlined in Chow et al. (2005c).

7 Acknowledgments

Textures used in the virtual environments were taken from Paul Bourke's texture library (Paul Bourke's Personal Pages).

The authors would like to thank all the volunteers who participated in this research experiment. This study has been approved by the Monash Standing Committee on Ethics in Research involving Humans (SCERH).

Yang-Wai Chow would also like to acknowledge the support of the Information and Communication Technologies (ICT) scholarship provided by the Victorian Government, Australia, and also the Monash Faculty of Information Technology scholarship granted for this research.

8 References

Barkans, A.C. (1997): High Quality Rendering using the Talisman Architecture. *Proc. of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, Los Angeles, California, 79-88.

Chalmers, A., Cater, K. and Mafioli, D. (2003): Visual Attention Models for Producing High Fidelity Graphics Efficiently. *Proc. of the 19th Spring Conference on*

Computer Graphics (SCCG), Budmerice Castle, Slovak Republic, 47-54.

Chow, Y.W., Pose, R. and Regan, M. (2005a): Large Object Segmentation with Region Priority Rendering. *Proc. of the 28th Australasian Computer Science Conference 2005 (ACSC 2005)*, Newcastle, NSW, Australia, 19-28.

Chow, Y.W., Pose, R. and Regan, M. (2005b): Region Warping in a Virtual Reality System with Priority Rendering. *Proc. of the 2nd IADIS International Conference on Applied Computing*, Algarve, Portugal, 451-458.

Chow, Y.W., Pose, R. and Regan, M. (2005c): Design Issues in Human Visual Perception Experiments on Region Warping. *Proc. of the 2nd IADIS International Conference on Applied Computing*, Algarve, Portugal, 210-217.

Ellis, S.R., Mania, K., Adelstein, B.D. and Hill, M. (2004): Generalizeability of Latency Detection in a variety of Virtual Environments. *Proc. Of the Human Factors and Ergonomics Society 48th Annual Meeting*. New Orleans, Louisiana, 2632.

Farell, B. and Pelli, D.G. (1999): Psychophysical Methods, or how to Measure a Threshold, and why. In *Vision Research: A Practical Guide to Laboratory Methods*. 129-136. CARPENTER, R.H.S. and ROBSON, J.G. (eds). Oxford, Oxford University Press.

Gescheider, G.A. (1985): *Psychophysics: Method, Theory and Application, 2nd Edition*. Hillsdale, New Jersey, Lawrence Erlbaum Associates.

Goldstein, E.B. (1999): *Sensation and Perception, 5th Edition*. Pacific Grove, California, Brooks/Cole Publishing Company.

Green, C.S. and Bavelier, D. (2003): Action Video Game Modifies Visual Selective Attention. *Nature*, **423**(29 May):534-537.

Levitt, H. (1971): Transformed Up-Down Method, *Journal of the Acoustical Society of America*, **49**(2): 467-477.

Mania, K., Adelstein, B.D., Ellis, S.R. and Hill, M.I. (2004): Perceptual Sensitivity to Head Tracking Latency in Virtual Environments with Varying Degrees of Scene Complexity. *ACM International Conference Proceeding Series, Proc. of the 1st Symposium on Applied Perception in Graphics and Visualization*. Los Angeles, California, **73**:39-47.

Mark, W.R., McMillan, L., and Bishop, G. (1997): Post-Rendering 3D Warping. *Proc. of the 1997 Symposium on Interactive 3D Graphics*, Providence, Rhode Island, 7-16.

Meehan, M., Razzaque, S., Whitton, M.C. and Brooks, F.P. (2003): Effect of Latency on Presence in Stressful Virtual Environments. *Proc. of IEEE Virtual Reality 2003*, Los Angeles, California, 141-148.

NVIDIA (2005): GPU Programming Guide version 2.4.0, NVIDIA Corporation, http://download.nvidia.com/developer/GPU_Programming_Guide/GPU_Programming_Guide.pdf. Accessed 5 August 2005.

Paul Bourke's Personal Pages: Texture Library. http://astronomy.swin.edu.au/~pbourke/texture/ Accessed 1 August 2005.

Plumert, J.M., Kearney, J.K. and Cremer, J.F. (2004): Distance Perception in Real and Virtual Environments. *Proc. of the 1st Symposium on Applied Perception in Graphics and Visualization (APGV)*, Los Angeles, California, 27-34.

Pose, R. and Regan, M. (1994): Techniques for Reducing Latency with Architectural Support. *Proc. of East-West International Conference on Multimedia, Hypermedia and Virtual Reality*, Moscow, Russia, 153-160.

Reddy, M. (1997): Perceptually Modulated Level of Detail for Virtual Environments. Ph.D. thesis. Dept. of Computer Science, University of Edinburgh, UK.

Regan, M. and Pose, R. (1993): An Interactive Graphics Display Architecture. *Proc. of the Virtual Reality Annual International Symposium (IEEE VRAIS '93)*, Seattle, Washington, 293-299.

Regan, M. and Pose, R. (1994): Priority Rendering with a Virtual Reality Address Recalculation Pipeline. *Proc. ACM SIGGRAPH '94, in Computer Graphics, Annual Conference Series*, Orlando, Florida, 155-162.

Regan, M.J.P., Miller, G.S.P., Rubin, S.M., and Kogelnik, C. (1999): A Real-Time Low-Latency Hardware Light-Field Renderer. *ACM SIGGRAPH '99, Proc. of the 26th Annual Conference on Computer Graphics,* Los Angeles, California, 287-290.

Torburg, J. and Kajiya, J.T. (1996): Talisman: Commodity Realtime 3D Graphics for the PC. *ACM SIGGRAPH '96, in Computer Graphics, Annual Conference Series*, New Orleans, Louisiana, 353-363.

Ulrich, R. and Miller, J. (2004): Threshold Estimation in Two-Alternative Forced-Choice Tasks: The Spearman-Karber Method. *Perception and Psychophysics*. **66**(3):517-533.

Watson, B.A., and Hodges, L.F. (1995): Using Texture Maps to Correct for Optical Distortion in Head-Mounted Displays. *Proc. of the IEEE Virtual Reality Annual Symposium (VRAIS '95)*, Research Triangle Park, 172-178.

Watson, B., Walker, N., Hodges, L.F. and Worden, A. (1997): Managing Level of Detail through Peripheral Degradation: Effects on Search Performance with a Head-Mounted Display. *ACM Transactions on Human-Computer Interaction*. **4**(4):323-346.

Willemsen, P., Colton, M.B., Creem-Regehr, S.H. and Thompson, W.B. (2004): The Effects of Head-Mounted Display Mechanics on Distance Judgments in Virtual Environments. *Proc. of the 1st Symposium on Applied Perception in Graphics and Visualization (APGV)*, Los Angeles, California, 35-38.

Wu, B., Ooi, T.L. and He, Z.J. (2004): Perceiving Distance Accurately by a Directional Process of Integrating Ground Information. *Nature*, **428**(4 March):73-77.

Rendering Multi-Perspective Images with Trilinear Projection

Scott Vallance **Paul Calder**

School of Informatics and Engineering
Flinders University of South Australia,
PO Box 2100, Adelaide, South Australia 5001,
Email: `Paul.Calder@flinders.edu.au`

Abstract

Non-linear projections of 3D graphical scenes can be used to compute reflections and refractions in curved surfaces, draw artistic images in the style of Escher or Picasso, and produce visualizations of complex data. Previously, most non-linear projections were rendered by ray tracing. This paper presents trilinear projection, a technique for rendering non-linear projections in a manner that achieves significant performance benefits by taking advantage of current rendering hardware and software.

Trilinear projections are geometrically similar to Phong-shaded triangular patches, and like Phong patches they can be joined to represent more complicated shapes. The paper details how a single trilinear projection projects a scene point, how projections of scene triangles can be built up by considering the connectivity of projected points, and how multiple trilinear projections can be combined. Finally, it outlines a method for using trilinear projection to approximate reflections and refractions on curved surfaces.

Keywords: Multi-Perspective Images, Non-Linear Projection, Reflections, Refractions, Rendering Algorithms.

1 Introduction

Most computer graphics rendering of 3D scenes is linear. For example, a perspective projection simulates the physics of optics, mapping scene data back to a single point in space as if viewed through a camera lens. Nonlinear projections differ from linear projections in that straight lines in 3D may not be straight lines when projected. Such projections can be used to compute reflections and refractions in curved surfaces, draw artistic images in the style of Escher or Picasso, and produce visualistations of complex data.

Previous techniques for nonlinear projection have used ray tracing or relied on distortion of the scene data. This paper presents a new technique, called trilinear projection, for rendering nonlinear projections in a manner analogous to perspective transformation matrix rendering. The technique is based on a trilinear interpolation similar to that used for Phong-shaded trianglular patches. And like Phong patches, multiple trilinear projections can be joined together to represent complex projection surfaces while maintaining continuity across sub-projections.

1.1 Nonlinear Projections

Nonlinear projections occur naturally as reflections and refractions on curved objects. The strange and distorted images seen in an amusement park mirror are a familiar example of the distortion nonlinear projections generate. These images have also been examined in art, photography and computer graphics where they have variously been named cubist images, multi-perspective images, multiple-centre-of-projection images and multi-perspective panoramas.

Traditional Chinese landscape paintings frequently contain different foci, or sub-images, which are seamlessly joined. For example, Figure 1 shows a scene in which the perspective shifts from left to right, following the path of the stream. German artist M. C. Escher frequently depicted views with multiple vanishing points, or perspectives. For example, Figure 2 has five different vanishing points: top left and right, centre, and bottom left and right. While the automatic generation of images like these from 3D geometry may not be practical, they illustrate the concept and the aesthetic potential.

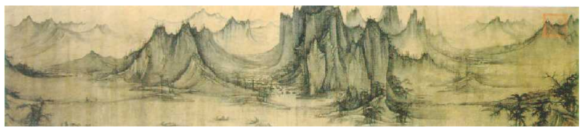


Figure 1: "Fishermans Evening Song" by Xu Daoning, circa 11th Century

Strip cameras are widely used in surveillance and mapping. These cameras have a continuous roll of film that slides past a slit as a picture is being taken. The camera may be moved whilst shooting, providing a change in point of view from one section of the film to another. For example, if used from a moving aeroplane a strip camera can capture a long section of curved earth as if it were flat. The technique has also been used for artistic purposes, such as the image in Figure 3 (Davidhazy 2001).

1.2 A Trilinear Projection Surface

In geometric terms, a perspective projection can be defined by a set of rays that emanate from a single point in space, and an orthographic projection by a set of parallel rays that emanate from a plane. By extending this approach, a nonlinear projection can be defined as a projection created by a set of rays that emanate from an arbitrary surface, with the origin and direction of each ray a function of the surface.

In computer graphics, complex curved surfaces are usually approximated as a mesh of triangles because triangle geometry is easy to compute and render. The

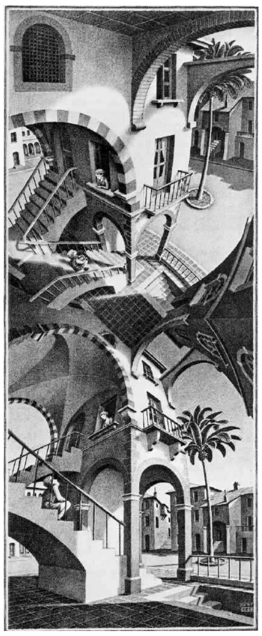


Figure 2: "High and Low" by M. C. Escher, an example of a nonlinear projection

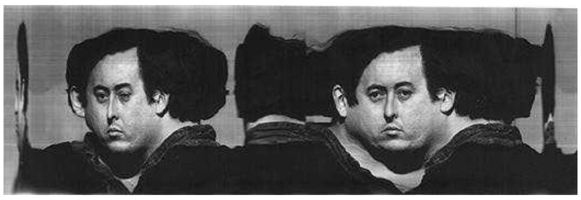


Figure 3: A strip camera photograph of a man's head

shape of a projection surface can thus be approximated by a suitable triangle mesh, and the directions of projection rays can be approximated by specifying the vertex normals of the triangles. The directions of rays internal to the triangles are defined implicitly by linear interpolation of the vertex rays.

Using this approach, a complex non-linear projection can be computed by tiling smaller projections – trilinear projections – each defined by a triangle with specified vertex positions and normals. Because in general the vertex normals of a projection triangle do not converge to a single point, a trilinear projection will be nonlinear. And because adjacent triangles share vertex normals, the combined projection will be continuous across triangle boundaries.

1.3 Related Work

Nonlinear projections can be rendered by ray tracing if a nonlinear projection can be expressed as a set of rays. For example, Löffelmann and Gröller (Löffelmann 1995) define an extended camera as a set of rays that start on a surface and point in the direction of the surface normal. The scene is then rendered with POVray [oVPL04], a widely available ray tracing implementation. Figure 4 shows a scene rendered with standard perpective projection and when rendered with a toroidal extended camera.

Rademacher and Bishop (Rademacher & Bishop 1998) describe techniques for generating multiple-centre-of-projection (MCOP) images. The images are generated by moving a virtual camera through a scene and capturing a single line of pixels at regular intervals. The technique is effectively a virtual strip camera, and can generate images such as the one in Figure 5.

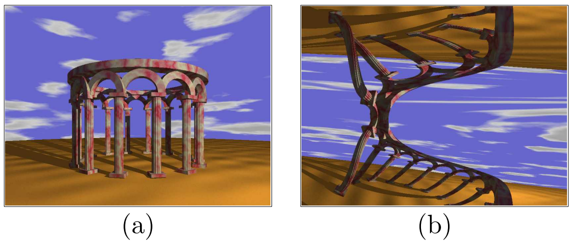


Figure 4: A scene of columns (a) rendered with conventional perspective (b) rendered from a torus surface

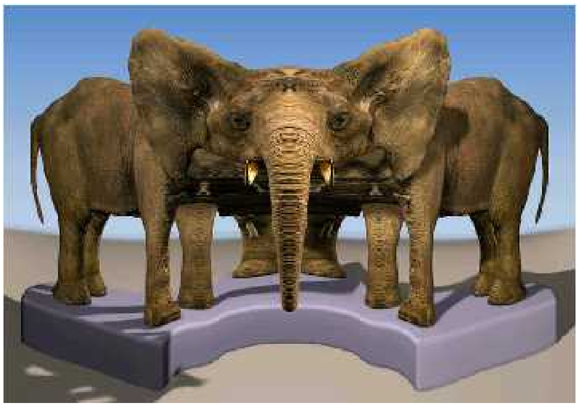


Figure 5: A nonlinear projection of an elephant

Yu and McMillan (Yu & McMillan 2004) introduce general linear cameras (GLC) as a mathematical description for a class of nonlinear projections defined by three rays passing through two parallel planes. The authors define and name various special cases of these cameras, and implement them with ray tracing and a light field rendering system. The technique is similar to trilinear projection but more constrained in that the vectors must have equal magnitude in the direction normal to the view. In that sense, GLC projections are a subset of trilinear projections.

Various authors have considered techniques for computing reflections from curved surfaces. For example, Glaeser (Glaeser 1999) presents equations for calculating the reflection of a space point on a sphere or cylinder of revolution, and environment mapping (originally described by Blinn and Newell (Blinn & Newell 1976)) approximates curved reflections by sampling the projection of a scene as if drawn from a point behind the reflection surface. Variations on environment mapping, such as extended environment maps (Cho 2000) and parameterised environment maps (Hakura, Snyder & Lengyel 2001), can produce more accurate images but at greater computational cost.

Ofek and Rappoport (Ofek & Rappoport 1999) describe an intriguing approximation for rendering reflections on curved surface that involves distorting objects based on the reflective surface so that they may then be rendered using standard perspective projection. The technique requires an appropriate tessellation of both reflective surface and scene object to approximate the curvature of the reflected lines. The correspondence between a point in space and the reflective surface is approximated by computing an explosion map (the projection of the reflective surface to the surface of a surrounding sphere) and then computing where on the map a scene point falls. The performance of the technique is sufficient for real-time rendering of moderate scenes, making it suited for visualisation tasks.

1.4 Organisation of the Paper

The remainder of this paper is organised as follows: Section 2 presents the algorithms and techniques for computing the trilinear projection of a single scene point. Section 3 describes how the projection for a scene triangle can be computed from the projections of its vertices. Since the projection is non-linear, and since each point can have up to 3 images, the projection for a single scene triangle can have up to 9 vertices and may consist of several disconnected shapes. Section 4 shows how multiple trilinear projections can be combined to represent projection from a complex surface. Section 5 outlines how trilinear projection can be applied to the task of computing reflections and refractions from curved surfaces. Finally, Section 6 briefly examines the performance characteristics of trilinear projection.

2 Projecting a Point

For purposes of this paper we define projecting a point as determining which location(s) on the projecting surface define a ray that intersects that point. A ray can be defined parametrically by a point, p and a normal n:

$$r(t) := p + tn \tag{1}$$

If p and n are defined using barycentric coordinates u and v, vertex positions $p_{1..3}$ and $n_{1..3}$, then a ray on the surface of the trilinear projection becomes:

$$\begin{aligned} r(u,v,t) &:= p(u,v) + tn(u,v) \\ &= (1-u-v)\,p_1 + up_2 + vp_3 + \\ &\quad t\left((1-u-v)\,n_1 + un_2 + vn_3\right) \end{aligned} \tag{2}$$

So for a scene point p_s:

$$\begin{aligned} p_s &= r(u,v,t) \\ &= (1-u-v)\,p_1 + up_2 + vp_3 + \\ &\quad t\left((1-u-v)\,n_1 + un_2 + vn_3\right) \end{aligned} \tag{3}$$

Solving for u, v and t in 3D means a system of three equations and three unknowns. The t and u, v terms are multiplied together making it a non-linear system of equations and an analytical solution is not readily apparent.

2.1 Treating the Trilinear Projection as a Parametric Triangle

To analytically solve this problem, instead of representing the surface as a set of rays, we represent it as a parametric triangle. Each vertex has a point and a normal associated with it. Treating these as rays we can extend along them according to the parameter t giving three new points, which form the vertices of a triangle as shown in Figure 6.

The three vertices of the parametric triangle are defined by the equations:

$$\begin{aligned} r_1 &:= p_1 + tn_1 \\ r_2 &:= p_2 + tn_2 \\ r_3 &:= p_3 + tn_3 \end{aligned} \tag{4}$$

A barycentrically defined point on the parametric triangle is:

$$\begin{aligned} p(u,v,t) &:= (1-u-v)\,r_1 + ur_2 + vr_3 \\ &= (1-u-v)\,(p_1 + tn_1) + \\ &\quad u\,(p_2 + tn_2) + v\,(p_3 + tn_3) \end{aligned} \tag{5}$$

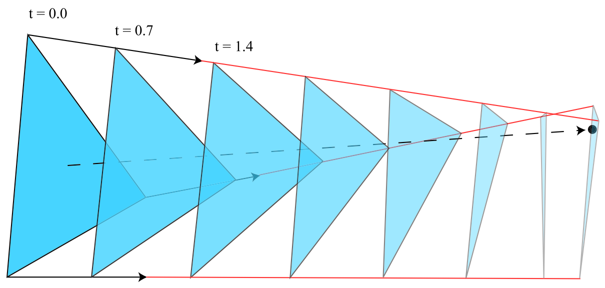


Figure 6: A parametric triangle shown at different values of t

The task of projecting a scene point now becomes that of finding a point $p(u,v,t)$ that coincides with the scene point. The u, v and t satisfying this constraint are the same as those satisfying $p_s = r(u,v,t)$ because the two equations are simply isomorphs.

The advantage in representing the triangle as a parametric triangle is that the parameter t can be determined independently of u and v. First, for the parametric vertices defined in Equation 4, find the values of t such that the vertices and the scene point p_s are coplanar, then solve for u and v in the plane of the parametric triangle.

2.2 Determining the Coplanarity of the Parametric Triangle and the Scene Point

Any four points can be considered a tetrahedron; four coplanar points form a tetrahedron whose volume is 0. For a tetrahedron defined by four points A, B, C and D the volume of the tetrahedron is:

$$volume := \frac{1}{4}\left|\mathbf{AB} \bullet (\mathbf{AC} \times \mathbf{AD})\right| \tag{6}$$

This equation can also be expressed as the magnitude of the determinant of the matrix containing the three vectors.

$$volume := \frac{1}{4}\left|\det\begin{bmatrix}\mathbf{AB}\\ \mathbf{AC}\\ \mathbf{AD}\end{bmatrix}\right| \tag{7}$$

Composing the three vectors from the parametric vertices defined in Equation 4 with the scene point p_s and substituting into Equation 7 gives the volume of the tetrahedron defined by those four points. When these points are coplanar this volume is 0. Removing the unnecessary constant and magnitude gives:

$$\begin{vmatrix} p_1 + tn_1 - p_s \\ p_2 + tn_2 - p_s \\ p_3 + tn_3 - p_s \end{vmatrix} = 0 \tag{8}$$

This can be expanded for three dimensions, giving a cubic polynomial in terms of t. Either one or three real solutions for t exist, and each value of t defines a potentially different projection of the scene point.

2.3 Precalculating Partial Coefficient Values

Computing the coefficients of Equation 8 involves substantial calculation not directly dependent on the scene point. These calculations depend only upon the values of the parametric triangle itself and therefore can be reused across scene points. This is useful because in a normal scene situation there are many scene points that need to be projected by each parametric triangle. The most common drawing primitive, the triangle, is comprised of three such scene points. The minor additional memory requirements of storing these values is small in comparison to the reduction in computation.

In Equation 7 the volume of a tetrahedron is defined as the determinant of three vectors formed from the four points. This can equally be expressed as the four by four determinant shown in Equation 9.

$$\begin{vmatrix} p_{sx} & p_{sy} & p_{sz} & 1 \\ p_{1x}+tn_{1x} & p_{1y}+tn_{1y} & p_{1z}+tn_{1z} & 1 \\ p_{2x}+tn_{2x} & p_{2y}+tn_{2y} & p_{2z}+tn_{2z} & 1 \\ p_{3x}+tn_{3x} & p_{3y}+tn_{3y} & p_{3z}+tn_{1z} & 1 \end{vmatrix} = 0 \qquad (9)$$

The determinant in Equation 9 can be expanded to Equation 10 where $E_{1..4}$ are each three by three determinants.

$$p_{sx}E_1 - p_{sy}E_2 + p_{sz}E_3 - E_4 = 0 \qquad (10)$$

Determinants $E_{1..3}$ are quadratics in t and E_4 is a cubic. This means the coefficient of t^3 depends only upon properties of the parametric triangle and not the scene point. Further, the other coefficients of the cubic defined by the full expansion of Equation 9 depend on the scene point in a useful way. Equations $E_{1..4}$ are defined by the general cubic equation, Equation 11, where $A_{1..3} = 0$

$$E_i = \quad A_i t^3 + B_i t^2 + C_i t + D_i, \quad i = 1,2,3,4 \, (11)$$

If the coefficients for the cubic equation defined by Equation 9 are represented by a, b, c and d, where $at^3 + bt^2 + ct + d = 0$, then the relationship between the coefficients and the properties $(A, B, C, D)_{1..4}$ can be described as:

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & A4 \\ B1 & B2 & B3 & B4 \\ C1 & C2 & C3 & C4 \\ D1 & D2 & D3 & D4 \end{bmatrix} \begin{bmatrix} p_{sx} \\ p_{sy} \\ p_{sz} \\ 1 \end{bmatrix} \qquad (12)$$

All properties $(A, B, C, D)_{1..4}$ can be calculated independently of scene points. The full derivations of $(A, B, C, D)_{1..4}$ are provided elsewhere (Vallance 2005). Using these pre-calculated partial values allows for faster calculation across multiple scene points.

3 Drawing a Scene Triangle

Each of the vertices of a scene triangle can be separately projected with the trilinear projection generating either one or three solutions for each vertex. The manner in which the projected vertices are connected can be understood by considering the sweep of the parametric triangle intersected with the scene triangle.

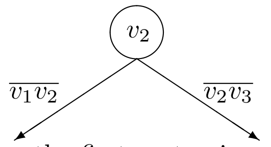
3.1 Computing the Projected Shapes

At every value of t the parametric triangle is a standard triangle in scene space. The intersection of this triangle's plane with the scene triangle's plane forms a line. As the value of t changes the intersection line traverses the plane of the scene triangle. The motion of this intersection line is continuous because t is continuous, except where the intersection line is undefined because the scene triangle and parametric triangle are parallel or coplanar.

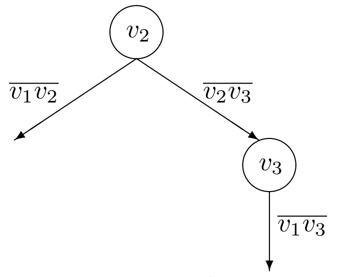
A straight line intersects at most two sides of a triangle. Furthermore, the locus of a continuously defined line must intersect a vertex on the end of an edge before intersecting the edge itself. The order, from smallest to largest t value, in which projected scene triangle vertices are intersected determines the connectivity of those vertices. Initially the line of intersection crosses the two edges connected with the vertex projected by the smallest t value. Each vertex thereafter toggles which edges are being intersected by the parametric triangle. When all edges are toggled off the line of intersection is no longer traversing the scene triangle and the projected shape is complete.

For strings of vertices, in order of t value, toggling the edge states reveals which vertices group together to form a shape. Even though the scene triangle has three vertices, it does not necessarily project shapes which have three vertices. Moreover, a single scene triangle can produce up to four shapes when projected with a parametric triangle. This particular case happens when the three scene vertices produce three t solutions each, and the nine projected vertices form three two-vertex shapes and one three-vertex shape.

For example, for a list of vertices $V = \langle v_2, v_3, v_3, v_1, v_3 \rangle$ which form a five-vertex shape the following diagrams show how these vertices are formed into a polygon. Each circle represents a processed vertex. An arrow between two circles represents the edge upon which those two vertics are connected. An unterminated arrow represents an open edge at that stage of the traversal. Only two arrows can be unterminated at a given stage in the traversal and the current unterminated arrows are named `left` and `right`.



Step 1: Since the first vertex is v_2 the two edges $\overline{v_1v_2}$ and $\overline{v_2v_3}$ become active. If the next vertex was a v_1 then it would be placed on `left` because it has to be connected to the edge $\overline{v_1v_2}$. Similarly if the next vertex was a v_3 it would be placed on `right`. Finally, as a v_2 would be connected to both edges, it would finish the shape, and could be placed on either `left` or `right`.



Step 2: The next vertex is in fact v_3 so it is placed on `right`. This toggles the active edges such that $\overline{v_2v_3}$ becomes inactive and $\overline{v_1v_3}$ active.

$$\begin{tikzcd} & v_2 \arrow[dl, "\overline{v_1 v_2}"'] \arrow[dr, "\overline{v_2 v_3}"] & \\ {} & & v_3 \arrow[d, "\overline{v_1 v_3}"] \\ & & v_3 \arrow[d, "\overline{v_2 v_3}"] \\ & & {} \end{tikzcd}$$

Step 3: The next vertex is also v_3, indicating a transition to the $\overline{v_2 v_3}$ edge. Accordingly the vertex v_3 is placed on the `right` and the `right`'s edge is changed.

$$\begin{tikzcd} & v_2 \arrow[dl, "\overline{v_1 v_2}"'] \arrow[dr, "\overline{v_2 v_3}"] & \\ v_1 \arrow[d, "\overline{v_1 v_3}"'] & & v_3 \arrow[d, "\overline{v_1 v_3}"] \\ {} & & v_3 \arrow[d, "\overline{v_2 v_3}"] \\ & & {} \end{tikzcd}$$

Step 4: In a similar way, vertex v_1 is placed on the `left` and the `left`'s edge is changed to $\overline{v_1 v_3}$.

$$\begin{tikzcd} & v_2 \arrow[dl, "\overline{v_1 v_2}"'] \arrow[dr, "\overline{v_2 v_3}"] & \\ v_1 \arrow[d, "\overline{v_1 v_3}"'] & & v_3 \arrow[d, "\overline{v_1 v_3}"] \\ v_3 & & v_3 \end{tikzcd}$$

Step 5: The final vertex is v_3 which could be placed in either list; it is arbitrarily placed on `left` for convenience. Previously active edges $\overline{v_1 v_3}$ and $\overline{v_2 v_3}$ are toggled off and the shape is finished.

3.2 Drawing the Projected Polygons

The result of projecting each vertex in a triangle and then assembling the vertices together is a set of polygons. One scene triangles may result in nine projected vertices, which can be arranged in up to four different polygons from two to nine vertices (though not all combinations are possible). Each projected vertex has a u, v and t value associated with it and these values correspond to post projection x, y and z coordinates. Using these coordinates a polygon can be drawn using standard rasterising techniques with visibility resolved by the depth coordinate. Rasterising assumes that the straight lines connect the projected vertices which is unlikely to be accurate. Figure 7 shows one scene triangle projected by a single trilinear projection resulting in two shapes, one with four vertices and the other with five. The left hand figure is the correct projection as computed by a ray tracing algorithm. The right hand figure shows the approximation introduced by trilinear projection.

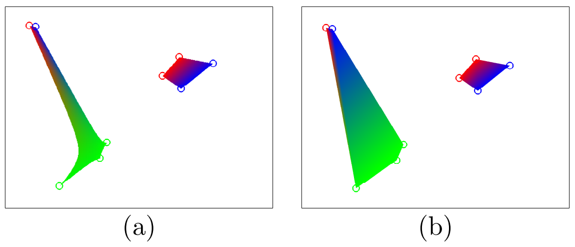


Figure 7: A projected triangle resulting in two shapes: (a) ray trace (b) trilinear projection

To more accurately represent the curves that connect projected vertices the solution can be tesselated. This can be done by sampling the scene triangle at intermediate t values. In the non-tesselated case the triangle is sampled at a value of t for each projected vertex. Extra values of t between the start and end of a shape can be inserted to increase the projected polygons accuracy.

For each t value the parametric triangle is a triangle in scene space. The intersection of this triangle and the scene triangle results in a line. For every value of t from the smallest for a particular shape to the largest the intersection line crosses the scene triangle. This line, clipped to the overlapping sections of the scene and parametric triangle (for this particular t value), corresponds directly to a line on the final image. Figure 8 shows a scene triangle rendered with 5 extra t samples per shape.

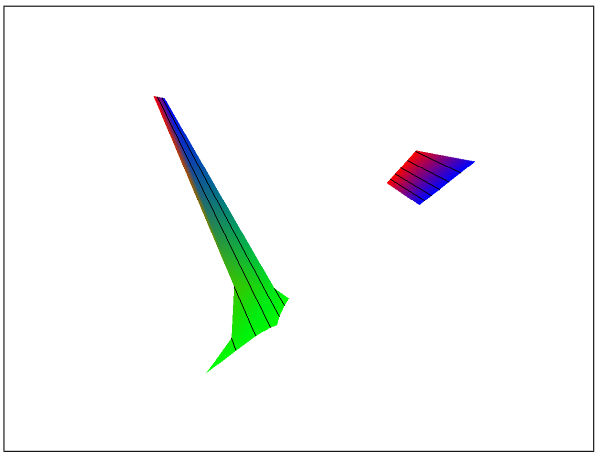


Figure 8: Scene triangle sampled at 5 extra t levels per shape approximating a (4,5) shape configuration

4 Combining Multiple Projections

A projection comprising a single trilinear projection has limited curvature. More complicated projections can be built with a mesh of trilinear triangles. In the same way that a triangular scene mesh can be approximated as smoothly curving by sharing surface normals, so can a projection mesh. When projecting with multiple trilinear triangles, issues of continuity arise that can be solved by clipping in scene space.

The simplest approach to the problem of clipping is to clip each projection in view space. This can result in discontinuity because the lines between projected vertices are drawn as straight when they are really curved. Discontinuities arising from this simplistic approach can be addressed by clipping scene triangles to the volume swept out by projecting the parametric triangle into scene space. The result is equivalent to tessellating the projected scene triangle at the boundary of the next trilinear projection. Each edge of a surface triangle forms a parametric line segment that traces out a surface through space which may intersect with edges in the scene triangles. To correctly clip to a parametric surface triangle region, the three parametric line segments defining the triangle edges must be traced through all the scene triangles, and the triangles clipped according to the intersections.

A line segment of the parametric triangle at t is defined parametrically by:

$$\begin{aligned} e\,(s,t) \;&:=\; p_i + tn_i + s\,(p_j + tn_j) \\ & \quad\; i,j = 1,2,3 \\ & \quad\; i \neq j \end{aligned} \tag{13}$$

where s varies from 0 to 1. Consider the intersection between a parametric edge, defined by the points $p_i + tn_i$ and $p_j + tn_j$, and a scene triangle edge, defined by the points p_{s1} and p_{s2}. The value of t at the intersection as projected out of the trilinear projection, can be determined independently of s because for the two line segments to intersect they must lie on the same plane. According to Equation 7 the four points are coplanar when:

$$\begin{vmatrix} p_{s1} - p_{s2} \\ p_i + t.n_i - p_{s2} \\ p_j + t.n_j - p_{s2} \end{vmatrix} = 0 \tag{14}$$

This can expanded giving a quadratic in terms of t. This quadratic may have two, or no real solutions. Each value of t defines a triangle and the scene line can be intersected with that triangle yielding a clipping point.

5 Applications

Non-linear projections have been explored in art and visualisation because of their ability to represent 3D objects in unusual ways. Though not as readily interpreted as perspective projections they may have the potential to illustrate complicated relations that are hidden in perspective projections. Finding non-linear projections that genuinely improve the task of visualisation is an unsolved problem. This paper instead examines the use of trilinear projection as way of approximating reflections and refractions on curved surfaces.

5.1 Reflection

The specular reflection of a ray is governed by the equation:

$$\theta_r = \theta_i \tag{15}$$

where θ_r is the angle of reflection and θ_i is the angle of incidence. This leads to following equation:

$$r = 2\,(n \cdot i)\,n - i \tag{16}$$

where r, n and i are unit vectors for the reflection, surface normal and incidence direction.

5.2 Refraction

Refraction is the bending of light due to the difference in refractive index between two materials. Simple refraction can be described by Snell's law, which shows the relation between angles of incidence, refraction and the refractive index of the materials. Snell's law is described by the following relation:

$$\frac{\sin\theta_i}{\sin\theta_t} = \frac{\eta_t}{\eta_i} \tag{17}$$

where θ_i is the angle of incidence, θ_t is the angle of transmission, η_t is the refractive index of the material the ray is entering and η_i is the refractive index of the material the ray is leaving. This leads to the following equation:

$$q = \eta i - (\cos\theta_t + \eta\cos\theta_i)n \tag{18}$$

where q, i and n are the vectors of transmission, incidence and the surface normal, and $\eta = \frac{\eta_t}{\eta_i}$.

5.2.1 Approximating Reflections and Refractions

For a given scene surface, represented by a polygon mesh with shared normal vectors (a Phong-shaded mesh), the reflected or refracted image in each segment can be approximated by a trilinear projection. The points $p_{1..3}$ and vectors $r_{1..3}$ in Figure 9 form a trilinear projection approximating the reflection seen from the eyepoint e in the triangle defined by the points $p_{1..3}$ and the normals $n_{1..3}$. The projection is exactly correct at the vertices, but the reflection vectors across the surface will in general only approximate the correct solution. This is because instead of interpolating the surface normal and then calculating the reflection vector, the reflection vector is interpolated from the vertex reflection vectors. In Figure 9 the reflection at point p should be along the ray r but in the trilinear projection it is along ray r'. Despite the inaccuracy the computed reflection has several desirable characteristics: it is nonlinear, which means that the reflection is curved, as is expected, and it is continuous across multiple reflective facets. Refractions, inter-reflections and inter-refractions can all be approximated by using the vector equations for reflection and refraction at the vertices of the face to generate a trilinear projection.

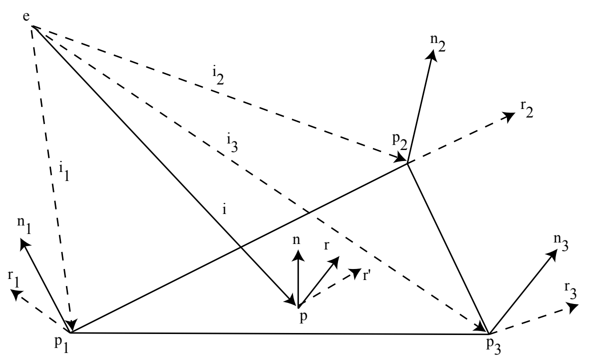


Figure 9: An approximation of a reflection with a trilinear projection

5.3 Example Projections

Figure 10 shows a scene with a reflective sphere and a cube that is being rendered from a view point close to the sphere's surface. The rays at the four corners of the view intersect the sphere and are reflected off at various angles. Figure 11 (a) shows a ray traced image similar to that which would be seen in the section of reflective sphere shown in Figure 10. Figures 11 (b) to (f) show a trilinear projection approximation of the image with increasing projection mesh resolution. In the 1x1 surface, Figure 11 (b), the trilinear projection is two coplanar triangles whose normals are the reflection vectors show in Figure 10. Figures 11 (c) to (f) are rendered from projection surfaces that are increasingly accurate approximations of the surface of the sphere and the reflection vectors off the sphere's surface. The increasing mesh resolution means more trilinear projections are used to approximate the reflection, resulting in more accurate rendering.

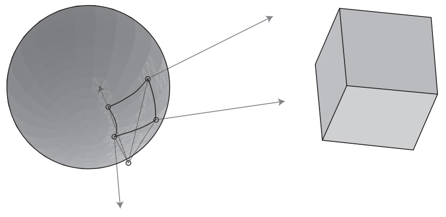


Figure 10: A cube reflected in a sphere

Figures 12 (a) to (f) show a refraction of a cube scene through a plane whose normals are coincident. This simulates the effect of viewing an object through a convex lens. Figure 12 (a) is the correct ray traced solution and figures 12 (b) through (f) show trilinear projections to approximate this using 1x1, 2x2, 3x3, 4x4 and 5x5 resolution meshes respectively.

6 Performance Characteristics

An implementation of the trilinear projection algorithms and a ray tracer was developed using OpenGL (Segal & Akeley 1998). The prototype can draw single and multiple trilinear projections, with or without clipping and tessellation. Reflective or refractive projection surfaces can be generated automatically, and the ray tracing implementation can render correct reflection and refraction solutions for comparison. The ray tracing code is naive, and performs no scene organisation optimisation to decrease the rendering time.

The results here were obtained on a 800 MHz Athlon with a GeForce 2mx graphics card. Execution speeds were averaged across multiple runs. Table 1 shows the speed in milliseconds to render a single frame of the scene presented in Figure 11 at resolutions of 800 × 800 pixels. The scene consists of 686 untextured triangles. Table 2 shows the time taken to render the images in Figure 12, also at 800 × 800 pixels with same scene rendered through a refractive sphere.

As noted before ray tracing can be speeded up by organising the scene so that for any particular ray only a subset of triangles need be intersected. Trilinear projection can also use scene organisation to improve speed because any particular trilinear projection may only image a subset of the scene. For a single trilinear projection there is only a linear performance cost over scanline rendering. This performance cost relates to determining the coefficients of

Image	Projections	Ray(ms)	Trilinear(ms)
(a)		177926.0	
(b)	2		24.4
(c)	8		70.1
(d)	18		150.1
(e)	32		230.2
(f)	50		360.7

Table 1: Execution time for rendering examples in Figure 11

Image	Projections	Ray(ms)	Trilinear(ms)
(a)		247576.0	
(b)	2		18.5
(c)	8		72.3
(d)	18		167.0
(e)	32		287.8
(f)	50		467.3

Table 2: Execution time for rendering examples in Figure 12

the cubic equation, solving the cubic and rejoining the projected vertices. The performance of rendering with multiple trilinear projections scales in a linear fashion.

7 Conclusion

The problem of rendering nonlinear projections with scanline-style algorithms has not previously been thoroughly examined. This paper presents an overview of a trilinear projection algorithm that projects scene points and triangles in a manner compatible with scanline rendering algorithms. The algorithms have been implemented and used to demonstrate a range of applications. More details of the algorithms and performance benchmarks appear elsewhere (Vallance 2005).

The performance characteristics of scanline rendering have made it very useful in interactive and animated computer graphics, and algorithms compatible with scanline rendering have a substantial base of hardware and software to draw upon. A limitation of scanline rendering has been its difficulty in modeling certain complicated optical interactions. The trilinear projection algorithm provides a new technique for rendering optical interactions that presents developers with more tools to render unusual visualisations and optical phenomena with acceptable performance.

References

Blinn, J. F. & Newell, M. E. (1976), 'Texture and reflection in computer generated images', *Communications of the ACM* **19**(10), 542–547.

Cho, F. (2000), Towards Interative Ray Tracing in Two- and Three-Dimensions, PhD thesis, University of California at Berkeley.

Davidhazy, A. (2001), 'Peripheral portraits and other strip camera photographs', Retrieved October, 2001 from the World Wide Web `http://www.rit.edu/~andpph/exhibit-6.html`.

Glaeser, G. (1999), 'Reflections on spheres and cylinders of revolution', *Journal for Geometry and Graphics* **3**(2), 121–139.

Hakura, Z., Snyder, J. & Lengyel, J. (2001), Parameterized environment maps, *in* 'Proc. of the 2001 Symposium of Interactive 3D Graphics'.

Löffelmann, H. (1995), Extended cameras for ray tracing, Master's thesis, Vienna Technical Institute.

Ofek, E. & Rappoport, A. (1999), Interactive reflections on curved objects, *in* 'Proc of SIGGRAPH 99'.

Rademacher, P. & Bishop, G. (1998), Multiple-center-of-projection images, *in* 'Proc. of SIGGRAPH 98'.

Segal, M. & Akeley, K. (1998), *The OpenGL Graphics System: a Specification (Version 1.2).*

Vallance, S. (2005), Trilinear Projection, PhD thesis, Flinders University.

Yu, J. & McMillan, L. (2004), General linear cameras, *in* T. Pajdla & J. Matas, eds, 'Computer Vision - ECCV 2004, 8th European Conference on Computer Vision, Prague, Czech Republic, May 11-14, 2004. Proceedings, Part II', Vol. 3022 of *Lecture Notes in Computer Science*, Springer.

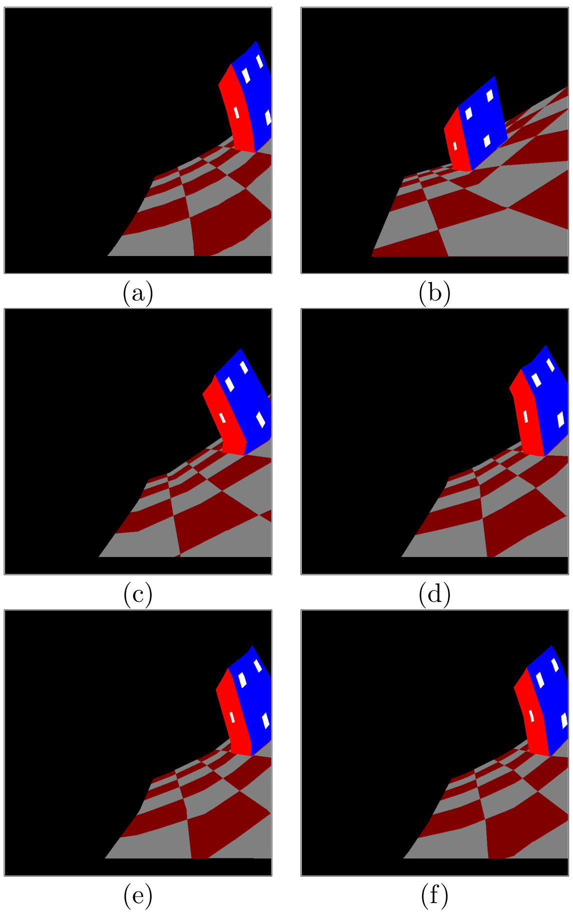


Figure 11: A cube reflected on a sphere: (a) ray traced, (b) 1x1 surface, (c) 2x2 surface, (d) 3x3 surface, (e) 4x4 surface, (f) 5x5 surface

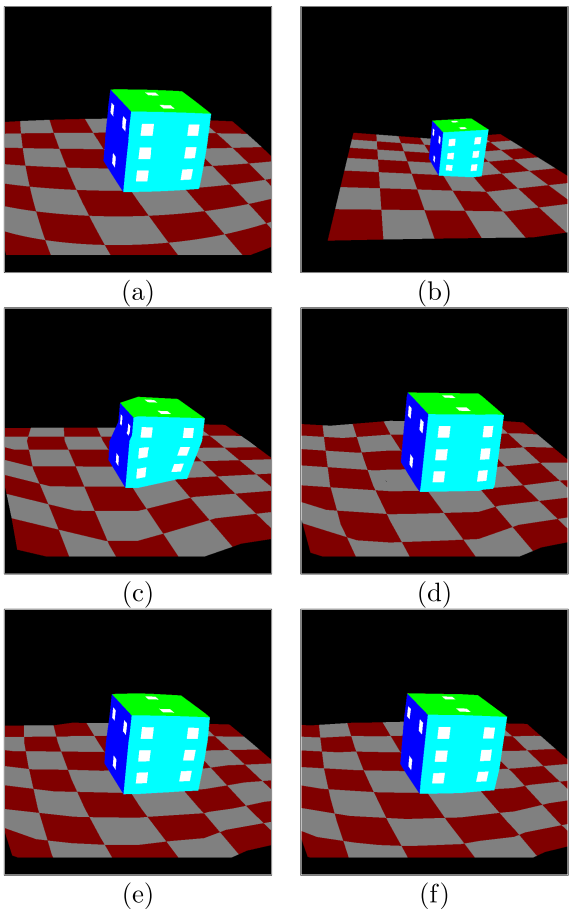


Figure 12: A cube refracted through a plane with spherical normals: (a) ray traced, (b) 1x1 surface, (c) 2x2 surface, (d) 3x3 surface, (e) 4x4 surface, (f) 5x5 surface

Extensible Detection and Indexing of Highlight Events in Broadcasted Sports Video

Dian W. Tjondronegoro[1], Yi-Ping Phoebe Chen[2], Binh Pham[3]

[1] School of Information Systems, Queensland University of Technology, Brisbane, Australia
[2] School of Information Technology, Deakin University, Melbourne, Australia
[3] Faculty of Information Technology, Queensland University of Technology, Brisbane, Australia

dian@qut.edu.au, phoebe@deakin.edu.au, b.pham@qut.edu.au

Abstract

Content-based indexing is fundamental to support and sustain the ongoing growth of broadcasted sports video. The main challenge is to design extensible frameworks to detect and index highlight events. This paper presents: 1) A statistical-driven event detection approach that utilizes a minimum amount of manual knowledge and is based on a universal scope-of-detection and audio-visual features; 2) A semi-schema-based indexing that combines the benefits of schema-based modeling to ensure that the video indexes are valid at all time without manual checking, and schema-less modeling to allow several passes of instantiation in which additional elements can be declared. To demonstrate the performance of the events detection, a large dataset of sport videos with a total of around 15 hours including soccer, basketball and Australian football is used.

Keywords: Extensible sports video indexing, multi-modal event detection

1 Introduction

Sports video indexing approaches can be categorised based on low-level (perceptual) *features* and high-level *semantic* annotation (Djeraba, 2002). There are some elements beyond perceptual level (known as the *semantic gaps*) which can make feature based-indexing tedious and inaccurate. For example, users cannot always describe the visual characteristics of certain objects they want to view for each query. In contrast, the main benefit of *semantic-based indexing* is the ability to support more intuitive queries. However, semantic annotation is generally time-consuming, and often incomplete due to the limitations of manual supervision and the currently available techniques for automatic semantic extraction. Therefore, video should be indexed using semantic that can be extracted automatically with minimal human intervention. Events-based indexing can be noted as the most suitable indexing technique for sport videos as sport highlights on TV, magazine or internet are commonly described using a set of events, particularly the important or exciting ones.

As there is yet a complete solution that can extract all events automatically, we need to design frameworks that support extensible detection and indexing of (highlight) events. Extensibility is emphasized as the algorithms developed for automatic extraction of features and semantic in sports video need to be extended gradually while improving the performance. As a result of more extractable contents, the indexing scheme needs to support continuous updates. The first and second section of this paper addresses each of these issues respectively. Following this, the experimental results that use a large dataset are reported before we close with some conclusions and future work.

2 Extensible Events Detection

It has become a well-known theory that sports events can be detected based on the occurrences of specific audio and visual features which can be extracted automatically. To date, there are two main approaches to fuse audio-visual features. One alternative, called *machine-learning* approach, uses probabilistic models to automatically capture the unique patterns of audio visual feature-measurements in specific (highlight) events. For example, Hidden Markov Model (HMM) can be trained to capture the transitions of *still*, *standing*, *walking*, *throwing*, *jumping-down* and *running-down* states during athletic sports' events (Wu et al., 2002). The main benefit of using such approach is the potential robustness, thanks to the modest usage of domain-specific knowledge which is only needed to select the best features set to describe each event. However, one of the most challenging requirements for constructing reliable models is to use features that can be detected flawlessly during training due to the absence of manual supervision. Moreover, adding a new feature into a particular model will require re-training of the whole model. Thus, it is generally difficult to build extensible models that allow gradual development or improvement in the feature extraction algorithms. To tackle this limitation, our statistical-driven models are constructed based on the characteristics of each feature. Any addition of a new feature will only result on the updates of the rules that were associated with that feature.

Another alternative for audio-visual fusion is to use manual heuristic rules. For example, the temporal gaps between specific features during basketball goal have a predictable pattern that can be perceived manually (Nepal et al., 2001). The main benefit of this approach is the absence of comprehensive training for each highlight and the computations are relatively less complex. However, this method usually relies on manual observations to construct the detection models for different events. Even though the numbers of domains and events of interest are limited and the amount of efforts is affordable, we primarily aim to reduce the subjectivity and limitation of manual decisions.

These two approaches still have two major drawbacks, namely, 1) the lack of a definitive solution for the scope of highlight detection such as where to start and finish the extraction. For example, Ekin et al (Ekin and Tekalp, 2003b) detect goals by examining the video-frames between the global shot that causes the goal and the global shot that shows the restart of the game. However, this template scope was not used to detect other events. On the other hand, Han et al (Han et al., 2003) used a static temporal-segment of 30-40 sec (empirical) for soccer highlights detection. 2) The lack of a universal set of features for detecting different highlights and across different sports. Features that best describe a highlight are selected using domain knowledge. For instance, whistle in soccer is only used to detect foul and offside, while excitement and goal-area are used to identify goal attempt (Duan et al., 2003).

In order to solve the first drawback, some approaches (Xu et al., 1998, Li and Ibrahim Sezan, 2001) have claimed that highlights are mainly contained in a *play* scene. However, based on a user study as reported in our earlier paper (Tjondronegoro et al., 2004b) , we have found that most users need to watch the whole play and break to understand fully an event. For example, when a *whistle* is blown during a *play* in soccer video, we would expect that something has happened. During the break, the *close-up views* of the players, a *replay scene*, and/or the *text display* will confirm whether it was a *foul* or *offside*. Consequently, it is expected that automated semantic analysis should also need to use both *play* and *break* segments to detect highlights. As for the second drawback, we aim to reduce the amount of manual choice of features set. For instance, it is quite intuitive to decide that the most effective event-dependent features to describe a *soccer foul* are *whistle*, followed by *referee appearance*. However, we were able to identify some additional characteristics of *foul* that could be easily missed by manual observation such as shorter *duration* (compared to shoot) and less *excitement* (compared to foul), based on statistical features that will be discussed in section 2.2.

2.1 Play-Break as Standard Scope of Events

Most broadcasted sport videos use transitions of typical shot types to emphasize story boundaries while aiding important contents with additional items. For example, a long global shot is normally used to describe an attacking play that could end with scoring of a goal. After a goal is scored, zoom-in and close-up shots will be dominantly used to capture players and supporters celebration during the break. Subsequently, some slow-motion replay shots and artificial texts are usually inserted to add some additional contents to the goal highlight. Based on this example, it should be clear that play-break sequences should be effective containers for a semantic content since they contain all the required details. Using this assumption, we should be able to extract all the phenomenal features from play-break that can be utilized for highlights detection. Thus, as shown in Figure 1, the scoping of highlight (event) detection should be from the last play-shot until the last break shot.

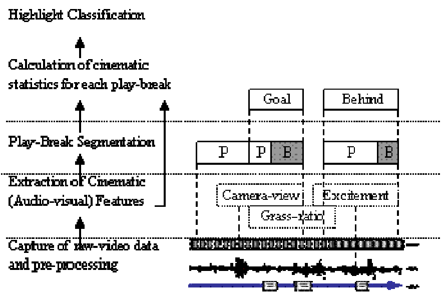

Highlight Classification
Calculation of cinematic statistics for each play-break
Play-Break Segmentation
Extraction of Cinematic (Audio-visual) Features
Capture of raw-video data and pre-processing
Goal
Behind
P
P
B
P
B
Camera-view
Excitement
Grass-ratio

Figure 1. Extracting Events from Play-Break.

Analysis of *camera-views* transition in a sports video has been used successfully for play-break segmentation (Ekin and Tekalp, 2003a). We have extended this approach by adding *replay-based correction* to improve the performance. Figure 2 shows how a replay scene (R) can fix the boundaries of play-break sequences – which are formed by a sequential play scene (P) and break scene (B). Please note that ".s" indicates start while ".e" indicates end. For example, R.s is short for the start of replay scene.

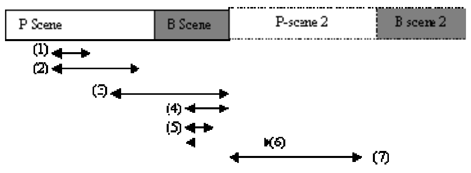


Figure 2. Locations of Replays in Play-breaks.

Based on these scenarios, an algorithm to perform replay-scene based play-break segmentation has been developed. This algorithm aims to: 1) fix the inaccurate boundaries of play-break sequences due to shorter breaks; 2) locate missing sequences due to missed breaks; and 3) avoid false sequences due to falsely detected play which is followed by a break.

Algorithm to fix play-break boundaries, based on replay scene locations

```
    If (A.s > B.s) & (A.e < B.e)
        A strict_during B
    If (A.s > B.s & A.e <= B.e) OR (A.s >= B.s & A.e < B.e)
        A during B
    If A.e = B.e
        A meets B
(1) If [R strict_during P] & [(R.e – P.e) >= dur_thres]
     B.s = R.s; B.e = R.e; Create a new sequence where [$P_2$.s = R.e+1] &
     [$P_2$.e P.e]
(2) If [R strict_during P] & [(R.e – P.e) <= dur_thres]
     P.e = R.e; B.s = R.e+1
(3) If [R meets B] & [R.s < P.e]
     P.e = R.s
(4-5) If [R during B] & [R meets B] ) OR (If [R strict_during B])
     No processing required
(6) If  [R during B] & [(R.e – $P_2$.s) >= dur_thres]
     B.e = R.e; Amend the neighbor sequence: [$P_2$.s = R.e+1]
(7) If [R during $P_2$] & [(R.e – $P_2$.s) >= dur_thres]
     Attach sequence 2 to sequence 1 (i.e. combine seq 1 and seq 2 into
     one  sequence)
```

It is important to note that some broadcasters insert some advertisements (*ads*) in-between or during the replay. To obtain the correct length of the total break, the total length of the ads has to be taken into account.

2.2 Statistical-Driven Events Detection

As most of the current cinematic-heuristics for highlight detection are heavily based on manual discoveries and domain-specific rules, we aim to minimize the amount of manual supervision in discovering the phenomenal features that exist in each of the different highlights. Moreover, in developing the rules for highlight detection, we should use as little domain knowledge as possible to make the framework more flexible for other sports with minimum adjustments. For this purpose, we have conducted a semi-supervised training from different broadcasters and different matches for each highlight to determine the characteristics of play-break sequences containing different highlights and no highlights. It is semi-supervised training as we manually classify the specific highlight that each play-break sequence contains. Moreover, the automatically detected play-break boundaries and mid-level features locations within each play-break such as excitement are manually checked to ensure the accuracy of training.

During training, statistics of each highlight are calculated with the following parameters (the examples are based on AFL video):

SqD = duration of currently-observed play-break sequence. For example, we can predict that a sequence that contains a goal will be much longer than a sequence with no highlight.

BrR = duration of break / *SqD*. Rather than measuring the length of a break to determine a highlight, the ratio of break segment within a sequence is more robust and descriptive. For example, we can distinguish goal from behind based on the fact that goal has a higher break ratio than behind due to a longer goal celebration and slow motion replay.

PlR = duration of play scene / *SqD*. We find that most non-highlight sequences have the highest play ratio since they usually contain very short break.

RpD = duration of (slow-motion) replay scene in the sequence. This measurement implicitly represents the number of replay shots which is generally hard to be determined due to many camera changes during a slow motion replay.

ExcR = duration of excitement / *SqD*. Typically, a goal consists of a very high excitement ratio whereas a non-highlight usually contains no excitement.

NgR = duration of the frames containing goal-area/duration of play-break sequence. A high ratio of near goal area during a play potentially indicate goal.

CuR = length of close-up views that includes crowd, stadium, and advertisements within the sequence / *SqD*. We find that the ratio of close-up views used in a sequence can predict the type of highlight. For example, goal and behind highlights generally has a higher close-up views due to focusing on just one player such as the shooter and goal celebration.

The statistical data of the universal feature sets within each highlight after a training that uses 20 samples is presented in Table 1. Based on the trained statistics, we have constructed a novel set of '*statistical-driven*' heuristics to detect soccer, AFL, and basketball highlights. We do not need to use any domain-specific knowledge, thereby making the approach less-subjective and robust when applied for similar sports. As each feature can be considered independently, more features can be introduced without the necessity to make major changes in the highlight classification rules. Moreover, our model does not need to be re-trained as a whole, thereby promoting extensibility. Hence, our approach will reap the full benefit when larger set of features are to be developed/improved gradually.

Highlight classification is performed as:

$$[\text{HgtClass}] = \text{Classify\_ Highlight}\ (D, NgR, Exc\ R, CuR, PlR,\ RpR)$$

where, HgtClass is the highlight class most likely contained by the sequence, while *D*, *NgR*, and so on are the statistical parameters described earlier. This equation will be performed according to the sport genre.

In order to classify which highlight is contained in a sequence, the algorithm uses some *measurements*. For example, in soccer, *G*, *S*, *F*, and *Non* are the highlight-score for goal, shoot, foul and non-highlight respectively. Each of these measurements is incremented by 1 point when certain rules are met. Thus, users should be able to intuitively decide the most-likely highlight of each sequence based on the highest score. However, to reduce users' workload, we can apply some post-processing to automate/assist their decision.

Feature	**Soccer** G=Goal, S=Shoot, F=Foul, N=Non_(avg; max; min)	**AFL** G=Goal, B=Behind, M=Mark, T=Tackle, N=Non_(avg; max; min)	**Basketball** G=Goal, F=Foul, FT=Free throw, T=Timeout_(avg; max; min)

Duration (D)	Gd_(73; 104; 43)
	Sd_(36, 73; 10)
	Fd_(38; 72; 14)
	Nd_(24; 40; 5)
		Gd_(72; 120; 40)
		Bd_(31; 53; 7)
		Md_(26; 65; 8)
		Td_(25; 63; 10)
		Nd_(20; 42; 8)
			Gd_(24; 51.6; 9.6)
			Fd_(28.8; 60; 12)
			FTd_(20.4; 30; 11)
			Td_(124.8; 255; 25)

Play Ratio (PlR)	Gp_(0.30; 0.46; 0.07)
	Sp_(0.57; 0.87; 0.15)
	Fp_(0.64; 0.97; 0.08)
	Np_(0.73; 0.91; 0.47)
		Gp_(0.17; 0.33;0.06)
		Bp_(0.38; 0.92; 0.10)
		Mp_(0.62; 0.86; 0.26)
		Tp_(0.55; 0.83; 0.08)
		Np_(0.52; 0.81; 0.17)
			Gp_(0.71; 0.94; 0.27)
			Fp_(0.48; 0.72; 0.13)
			FTp_(0.50; 0.81; 0.23)
			Tp_(0.12; 0.24; 0.05)

Near Goal (NgR)	Gn_(0.47; 1; 0.13)
	Sn_(0.55; 0.93; 0)
	Fn_(0.23; 0.81; 0)
	Nn_(0.17; 0.1; 0)
		Gn_(0.13; 0.43; 0.02)
		Bn_(0.10; 0.39; 0.02)
		Mn_(0.02; 0.23; 0)
		Tn_(0.01; 0.05; 0)
		Nn_(0.01; 0.08; 0)
			Gn_(0.49; 0.92; 0.04)
			Fn_(0.43; 0.93; 0)
			FTn_(0.55; 1; 0.05)
			Tn_(0.34; 0.85; 0)

Excitement (ExcR)	Ge_(0.45; 0.83; 0.10)
	Se_(0.35; 0.79; 0)
	Fe_(0.20; 0.50; 0)
	Ne_(0.2;0.6; 0)
		Ge_(0.29; 0.54; 0)
		Be_(0.38; 0.86; 0)
		Me_(0.32; 0.91;0)
		Te_(0.22; 0.59; 0)
		Ne_(0.30; 0.75; 0)
			Ge_(0.41; 0.82; 0.05)
			Fe_(0.34; 0.78; 0)
			FTe_(0.44; 0.90; 0)
			Te_(0.24; 0.43; 0.05)

Close-up (CuR)	Gc_(0.26; 0.51; 0.08)
	Sc_(0.23; 0.74; 0)
	Fc_(0.12; 0.29; 0)
	Nc_(0.2; 0.6; 0)
		Gc_(0.35; 0.86; 0)
		Bc_(0.35; 0.76; 0)
		Mc_(0.28; 0.56; 0)
		Tc_(0.18; 0.44; 0)
		Nc_(0.29; 0.69; 0)
			Gc_(0.11; 0.3; 0)
			Fc_(0.27; 0.69; 0)
			FTc_(0.26; 0.68; 0)
			Tc_(0.49; 0.78; 0.16)
			Nc_(0.2; 0.63; 0)

Replay (RpD)	Gr_(25; 34; 20)
	Sr_(6; 16; 0)
	Fr_(6; 23; 0)
	Nr_(0; 0; 0)
		Gr_(9; 23; 0)
		Br_(6; 40; 0)
		Mr_(1; 14;0)
		Tr_(4; 14; 0)
		Nr_(0; 0; 0)
			Gr_(0; 0; 0)
			Fr_(4.8; 13; 0)
			FTr_(0; 0; 0)
			Tr_(16; 40; 0)

Table 1. Statistics of Soccer, AFL, and Basketball Highlights.

The essence of highlight classification is on comparing the value of each input parameter against the typical statistical characteristics: min, avg, and max which are denoted as a *stat*. The following algorithm describes the calculation that can be applied to any sport (using soccer as an example).

Common event classification algorithm

```
Let $\text{Det\_Soccer\_Region}(val) = \text{Region}(val, \text{stat}_G, stat_S, stat_F, stat_N)$

Perform
$\text{region}_{1..n} = \text{Det\_Soccer\_Region}(D),(NgR),(ExcR),(CuR),(PlR),(RpR)$
For region$_1$ to region$_n$
Increment the corresponding highlight score //G, Sh, F, Non in this case
```

where,

$$\text{Region}(val,\ stat_1,\ stat_2,\ ...\ stat_n) = \begin{cases} 1,\ \text{if } (AvgD_1 \le MinAvgD)\,\&\,(TD_1 \le MinTD) \\ 2,\ \text{if } (AvgD_2 \le MinAvgD)\,\&\,(TD_2 \le MinTD) \\ ... \\ n, \text{if } (AvgD_n \le MinAvgD)\,\&\,(TD_n \le MinTD) \end{cases}$$

$\text{stat}_n = \{\text{avg}_n, \text{min}_n, \text{max}_n)$, $AvgD_n = \left|val - stat_n^{avg}\right|$,

$TD_n = \left|val - stat_n^{\max}\right| + \left|val - stat_n^{\min}\right|$,

$MinAvgD = \min(\text{AvgD}_1, AvgD_2, ...AvgD_n)$, $MinTD = \min(\text{TD}_1, TD_2, ...TD_n)$.

It is to be noted that in Det_soccer_region(*val*), stat_X matches the value input. Therefore, when *val* is *NgR*, then stat_G = {Gn_avg, Gn_max, Gn_min} is used according to the statistics-table.

In addition to the common algorithm, we can improve the accuracy of the event classification for a particular sport based on its statistical phenomena. This concept is described in the rest of this section.

2.2.1 Events Classification in Soccer

When play ratio, sequence duration and near goal ratio fall within the statistics of goal or shoot, it is likely that the sequence contains goal or shoot. Otherwise, we will usually find a foul or non-highlight. However, shoot often has similar characteristics with foul. In order to differentiate *goal* from *shoot*, and *shoot/foul* from *non-highlight*, we apply some statistical features:

> *Goal vs. Shoot:* Compared to shoot, goal has longer duration, more replays and more excitement. However, goal has shorter play scene due to the dominance of break during celebration.
>
> *Shoot, Foul, vs. Non-highlight (None):* None does not contain any replay whereas foul contains longer replay than shoot in average. Foul has the lowest close-up ratio as compared to shoot and none. None has the shortest duration as compared to shoot and foul. None contains the least excitement as compared to shoot and foul, whereas foul has less excitement than shoot.

Based on these findings, the following algorithm is developed.

Specific algorithm to classify highlight events in soccer

```
Perform $\text{region}_{1..3}$ = Det_Soccer_Region (PlR), (D), (NgR) accordingly
If all region1, 2 and 3 = 1 or 2
    //Most likely to be goal or shoot
    Increment G and Sh
    Perform $\text{region}_{4..7} = \text{Det\_Soccer\_Region}(ExcR), (RpD), (PlR), (D)$
    For region4 to region7
        If current region = 1, Increment G
        Else if current region = 2, increment Sh
        Else
            //Most likely to be foul, shoot, or non
            Increment F, Sh, Non
            Perform $\text{region}_{4..7} = \text{Det\_Soccer\_Region}\ (CuR), (ExcR), (D), (RpD)$
    For region4 to region7
        If current region = 2, increment Sh
        Else if current region = 3, Increment F
        Else if current region = 4, increment Non
```

It should be noted that the more compact representation of this algorithm is presented in Figure 3, where {val} is the convention of $\text{region}_{1..N} = \text{Det\_Soccer\_Region}(\text{val}_1), (\text{val}_2),..(val_N)$. Thus, squares denote the statistics that need to be checked, whereas the non-boxed texts are the associated highlight point(s) that will be incremented based on the outputs of each region. This representation is used for describing other sports.

2.2.2 Events Classification in AFL

In AFL, a *goal* is scored when the ball is kicked completely over the *goal-line* by a player of the attacking team without being touched by any other player. A *behind* is scored when the football touches or passes over the goal post after being touched by another player, or the football passes completely over the behind-line. A *mark* is taken if a player catches or takes control of the football within the playing surface after it has been kicked by another player a distance of at least 15 meters and the ball has not touched the ground or been touched by another player. A *tackle* is when the attacking player is being forced to stop from moving because being held (tackled) by a player from the defensive team. Based on these definitions, it should be clear that goal is the hardest event to achieve. Thus, it will be celebrated longest and given greatest emphasis will be given by the broadcaster. Consequently, behind, mark and tackle can be listed in the order of its importance (i.e. behind is more interesting than mark).

Figure 4 shows the highlight classification rules for AFL. Let *G, B, M, T, Non* be the highlight-score for *goal*, *behind*, *mark*, *tackle* and *non-highlight* respectively. Thus, for AFL event detection:

$\text{Det\_AFL\_Region}(val) = \text{Region}(val, \text{stat}_G, stat_B, stat_M, stat_T, stat_N)$.

The algorithm firstly checks that if current *PlR* belongs to stat_G (i.e. output = 1) and *NgR* is greater than the minimum of the typical value for goal and behind, then the sequence is most likely to contain either goal or behind. This is followed by comparing: *ExcR*, *RpD*, and *PlR* values: the outputs determine which score is incremented from *G* or *B*.

Else (if *PlR* does not belong to stat_G), it is more likely to contain mark, tackle, or none. This is followed by comparing: *D*, *CuR*, *PlR*, and *RpR* values: the outputs determine which score is incremented from *M*, *T*, or *N*.

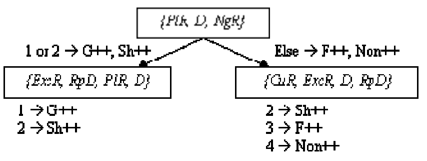


Figure 3. Highlight Classification Rules for Soccer

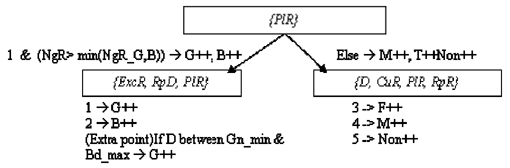


Figure 4. Highlight Classification Rules for AFL.

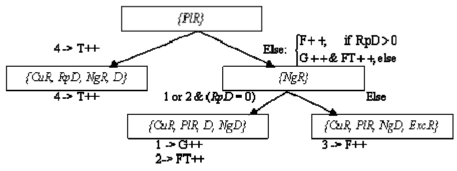


Figure 5. Highlight Classification Rules for Basketball

2.2.3 Events Classification in Basketball

Compared to soccer and AFL, goals in basketball are not celebrated and do not need a special resume such as kick off. Therefore, it is noted that the rules applied to soccer and AFL cannot be used directly for basketball goals.

Figure 5 shows the highlight classification rules for basketball. Let *G*, *FT*, *F*, *T* be the highlight-score for *goal*, *free-throw*, *foul*, and timeout respectively. Thus, for basketball event detection, let: $\text{Det\_Basketball\_Region}(val) = \text{Region}(val, \text{stat}_G, stat_{FT}, stat_F, stat_T)$

The algorithm firstly checks if current *PlR* belongs to stat_T (i.e. output = 4), then the sequence is most likely to contain timeout. This is followed by comparing: *Cur*, *RpD*, *NgR*, and *D* values: each time that the output of comparison is equal to 4, *T* is further incremented.

Else (if current *PlR* does not belong to stat_T), it is more likely to contain goal, free-throw, or foul (if RpD > 0). This is followed by checking:

If NgR belongs to region stat_G or stat_{FT} (i.e. output = 1 or 2), then the comparison is based on the values of: *CuR*, *PlR*, *D*, and *NgD*: the outputs determine which score is incremented from G or FT.

Else, (if NgR does not belong to region stat_G or stat_{FT}), then the comparison is based on the values of: *CuR*, *PlR*, *NgD*, and *ExcR*: each time that the output of comparison is equal to 3, F is further incremented.

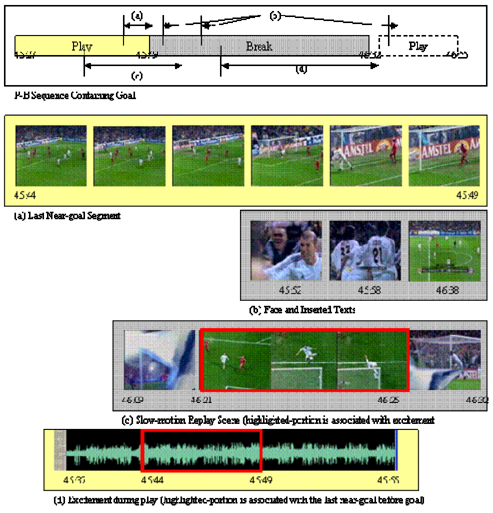
3 Extensible Indexing

For the indexing of events, OO modeling is recognized for its ability to support complex data definitions. We have identified two main alternatives in using O-O for modelling data based on the models presented in AVIS (Adali et al., 1996) and OVID (Oomoto and Tanaka, 1997), namely, schema-based and schema-less,. A schema-based model (Adali et al., 1996) can be composed of three types of *entities* (i.e. index-able items) in a video database, namely, 1) *video objects*, which capture entities that present in the video frames, 2) *activity* types, which is the subject of a frame sequence, and 3) *event*, which is the instantiation of an activity type. Thus, their model has allowed users to query the location of the occurrence of their desired object or events. The main benefit of using a schema-based model is its capability to support easy updates due to the strict components that have to be followed exactly for each entity. However, the main limitation is the difficulty to include new description during instantiation of video models due to the static schema; therefore, the model is not extensible.

In contrast, schema-less modeling (Oomoto and Tanaka, 1997) is designed based on the fact that each video interval can be regarded as a video object, in which the attributes can be objects, events, or other video objects. Thus, the content of a video object is more dynamic. Moreover, they also proposed dynamic calculation of inheritance, overlap, merge and projection of intervals to satisfy user queries. However, there are two main problems of schema-less modelling. First, query difficulties arise as users/developers must inspect the attribute definition of each object to develop a query because each object has its own attribute structure. Second, the total dependency on users or applications for supervising the instantiation of video objects occurs due to the fact that a schema is not present.

In order to combine the strengths of schema-based and schema-less modelling, this section demonstrates the utilization of XML to design and construct a semi-schema based video model. Schema-based matching ensures that the video indexes are valid during data operations such as insertion, thereby minimizing the need of manual checking. However, the model is also semi-schema based as it allows additional declared elements in the instantiated objects as compared to its schema definition. Moreover, not all elements in an object need to be instantiated at one time as video content extraction often requires several passes due to the complexity and lengthy processing; thereby supporting an extensible modeling scheme.In addition to the strength of OO modeling, the video model also attempts to benefit from relational modeling scheme. In particular, the utilization of *referential integrity* (Connolly and Begg., 2002) allows an object to include elements which are referenced from the existing objects within the database. The main purpose is to reduce objects being added within another object(s), thereby avoiding complex hierarchies and potential redundancies. Hence, in overall, the proposed video model supports object-relational modeling approach while adopting semi-schema based index construction and maintenance.

The sport video indexing is designed using two main abstraction classes, namely, *segment* and *event*. Each segment is instantiated with a unique key of segment Id into either: video-, visual-, or audio-segment. A segment, as shown in Figure 7, can be instantiated as video-, audio- or visual-segment which are extracted from a raw video track when mid-level features (e.g. whistle and excitement) can be detected. An event can be instantiated into generic (e.g. interesting event), domain-specific (e.g. soccer goal), or further-tactical (e.g. soccer free kick) semantics. Events and segments are chosen as they can provide an effective description for many sport games. For example, most users will benefit from watching soccer goals as the most celebrated and exciting event. Segments are used as the text-alternative annotations to describe the goal. As shown in Figure 6, the last near-goal segment in a play-break sequence containing goal describe *how* the goal was scored. Face and text displays can inform *who* scored the goal (i.e. the actor of the event) and the updated score. Replay scene shows the goal from different angles to *further emphasize* the details of how the goal is scored. In most cases, when the replay scene is associated with excitement, the content is more important. Excitement during the last play shot in a goal is usually associated with descriptive narration about the goal. In fact, we (human) often can hear a goal without actually seeing it.



(a)
(b)
Play
Break
Play
(c)
(d)
P-B Sequence Containing Goal
45:44
45:49
(a) Last Near-goal Segment
45:52
45:58
46:38
(b) Face and Inserted Texts
46:09
46:21
46:25
(c) Slow-motion Replay Scene (highlighted-portion is associated with excitement
45:44
45:49
45:55
(d) Excitement during play (highlighted-portion is associated with the last near-goal before goal)

Figure 6. Goal Event with Segment-Based Annotations.

We have utilized some of the main benefits from using XML to store and index the extracted information from sport videos:

XML is extensible by allowing additional information without affecting others. This is important to support gradual developments of feature extraction techniques that can add extractable segments and events.

XML is internally descriptive and can be displayed in various ways. This is important to allow users browsing the XML data directly, while search results can also be returned as XML that can provide direct link(s) to the video location.

XML fully supports semi-structured aspects that match video database characteristics: 1) Object can be described using attributes (properties), other objects (i.e. nested object), or heterogeneous elements (i.e. *any* element). Instantiated objects from the same class may not have the same number of attributes as not all attributes are compulsory, depending on the min and max occurs. 2) XML supports two types of relationships: nesting and referencing. However, to reduce redundancy, we have used referencing instead of nested object class.

We have used *XML Schema* to define and construct the XML-based video schema as it has replaced *DTD* as the most descriptive language. Due to its expressive power, XML schema has also been used as the basis of MPEG-7 DDL (Data Definition Language) and XQuery data model. Therefore, we should be able to easily leverage our proposed model to support MPEG-7 standard multimedia descriptions and XQuery implementation. For a more compact representation of XML schema, this section will demonstrate the use of *ORA-SS* (*Object-Relationship-Attribute notation for Semi-Structured data*) (Dobbie et al., 2000) to design the video model as shown in Figure 7 to Figure 9 (that is located on the last page). ORA-SS notation is chosen for its ability to represent most of XML schema's features. It is to be noted that our diagrams extend the ORA-SS notation by demonstrating a more complex sample which integrate inheritance diagram with schema diagram. We have also introduced two additional notations: 1) *italic texts* indicate abstract object, 2) ◸ (in Figure 9) indicates repeated object to avoid complex/crossing lines.

The followings describe the overall video indexing model. As shown in Figure 9, a *sport video* (*SV*) is a type of video segment which consists of SV components, overall summary, and hierarchical summary. *SV components* are composed of: 1) *segment collection* which stores a flat-list of audio, visual and audio segments that can be extracted from the sport video, 2) *syntactic relation collection* which stores all the syntactic relations such as 'composed of' and 'starts after' between one source segments and one or more destination segments, and 3) *semantic relation collection* which records all the semantic relations such as 'is actor of' and 'appears in' between one source segment or semantic object and one or more destination segments or semantic objects. *Overall summary* describes the sport video game as a whole; it includes where (stadium), when (date time), who (teams that compete), final result, and match statistics. Match statistics can be stored as XML tags or a visual frame such as text displays that depicts the number of goals, shots, fouls, red/yellow cards, and counter attacks in a soccer game. *Hierarchical summary* is composed of *comprehensive summary* and *highlight events* (*HE*) summary. *Comprehensive* summary describes sport video in terms of play-break sequences which are the main story decomposition unit in most of sport videos. For example, an attacking attempt during a play is stopped when there is a goal or foul. Each play-break can contain zero or one (key) event and can be decomposed into one or more play and break shots. Each play or break can be described by text-alternative annotations, including face, replay and excitement which are referenced (segments) from segment collection. On the other hand, *HE* summary organizes highlight events into common summary theme such as soccer goals and basketball free throws.

Each time sport video is instantiated, it will be specialized into the classified genre, such as soccer video, basketball video and AFL video. Therefore, a *soccer video* will inherit all components of (general) sport video while providing extra attributes such as *sport category* and some extra components. In particular, for each type of sport video, we can extract *domain specific events* such as soccer goal. Each domain event can be described using specific roles such as goal scorer. It should be noted that goal scorer will reference to a player that is defined elsewhere in order to avoid nested components. Similarly, domain events are referenced by hierarchical summaries. Finally, a *sport video database* is composed of one or more classified *sport videos*, and one *semantic object collection*. Semantic object collection defines the details of all the semantic objects that appear in the sport videos. For example, player can be instantiated into soccer player which is described by the specific attributes of a soccer player such as squad number, and preferred position.

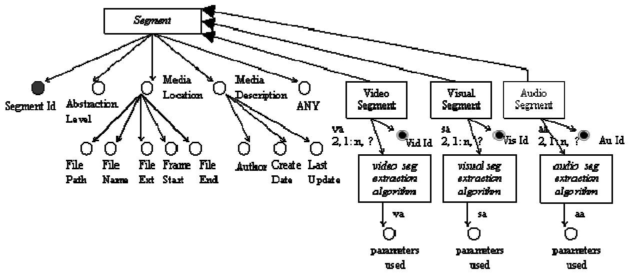


Figure 7. Extensible Indexing Scheme (1).

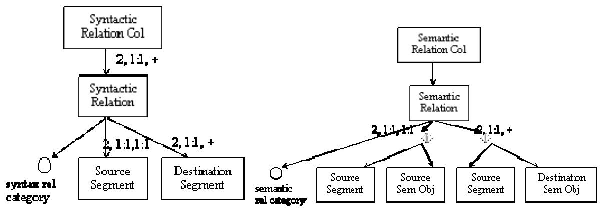


Figure 8. Extensible Indexing Scheme (2).

It is to be noted that in order to achieve a faster gradual index construction, all segments should be able to be extracted incrementally in the same level, without concerning about the hierarchy. For example, assuming that *PB1* contains *P1*, *P2*, and *B1*, the system should be able to add *B1* without necessarily attaching it to *PB1*. This allows the system to easily add *P1* and *P2* at later time. Therefore, hierarchy structures should be stored separately as a *hierarchical view* or processed dynamically when required by users for browsing.

Using the proposed video model, we have demonstrated a sport video indexing scheme that supports:

Extensible video indexes that allow gradual extraction of segments and events without affecting the others. For example, we can introduce more segments and events incrementally without affecting the existing ones. Similarly, more semantic objects, such as stadium and referee, can be introduced at a later stage when many sport videos share the same stadium and referee.

Object-Relationship modeling scheme. In particular, we have demonstrated that inheritance and referencing are important features in video database modeling. Inheritance enables us to reuse existing parent components while refining them with more specific items. Referencing enables us to store video components into a flat list which can be referenced by hierarchical structures to avoid redundancies.

Semi-schema based modeling scheme. As shown in Figure 7, we allow users/applications to add *ANY* additional elements (or attributes) into a segment description as long as the element has been declared somewhere else in the proposed schema, or other schema within a particular scope. In fact, we may attach ANY into other elements in our data model to allow more flexibility as users often know better what they want to describe than developers. However, we aim gradually modifying the schema with new components, especially when the extra information provided by users can be used to enrich the current video model.

4 Experimental Results

Performance results for mid-level features extraction (that are required during training and evaluation) including view classification, near-goal, and excitement, have been presented in our previous papers (Tjondronegoro et al., 2004a). For AFL and basketball videos, we only need to ensure that the adaptive thresholds are effective for each video sample. For this purpose, we compare the truth and the automatic results of features detection on each video for duration of 5-10 minutes. We then select the best empirical thresholds that can be applied to all videos within the same domain. Missing and/or false detections on individual mid-level features detection have less significant impacts on the highlights classification as the models depend on the fusion of all features. For example, soccer goal will still be detectable even if the near goal ratio and excitement is not detected perfectly. Nevertheless, the more accurate mid-level features can be extracted, the highlight points will be more accurately calculated. Hence, during experiment we have set a minimum value that highlight point should reach to be trusted. For all sport videos, we have successfully applied a minimum of 3 points for all highlights which means that at least 3 mid-level features can be detected. In almost all cases, highlights can be detected with a 6 to 7 point minimum threshold.

Table 2 will describe the video samples used during experiment. For each sport, we have used videos from different competitions, broadcasters and/or stage of tournament. The purpose is, for example, final match is expected to contain more excitement than a group match while exhibition will show many replay scenes to display players' skills. Our experiment was conducted using MATLAB 6.5 with image processing toolbox. The videos are captured directly from a TV tuner and compressed into '.mpg' format which can be read into MATLAB image matrixes.

Sample Group (Broadcaster)	**Videos "team1-teams2_period-[duration]"**
Soccer: UEFA Champions League Group Stage Matches (SBS)	*ManchesterUtd-Deportivo1,2-[9:51, 19:50]* *Madrid-Milan1,2[9:55,9:52]*
Soccer: UEFA Champions league (SBS)	*Juventus-Madrid1,2:[19:45,9:50]* *Milan-Internazionale1,2:[9:40,5:53]*

Elimination Rounds	Milan-Depor1,2-[51:15,49:36] **(S1)** Madrid-BayernMunich1,2-[59:41,59:00] **(S2)** Depor-Porto-[50:01,59:30] **(S3)**
Soccer: FIFA World cup Final (Nine)	*Brazil-Germany [9:29,19:46]*
Soccer: International Exhibition (SBS)	Aussie-SthAfrica1,2-[48:31,47:50] **(S4)**
Soccer: FIFA 100th Anniversary Exhibition (SBS)	Brazil-France1,2-[31:36,37:39] **(S5)**
AFL League Matches (Nine)	COL-GEEL_2-[28:39] **(A3)** StK-HAW_3-[19:33] **(A4)** Rich-StK_4-[25:20] **(A5)**
AFL League Matches (Ten)	COL-HAW_2-[28:15] **(A1)** ESS-BL_2-[35:28] **(A2)** BL-ADEL_1,2:[35:33,18:00] **(A6)**
AFL League Final rounds (Ten)	Port-Geel_3,4-[30:37,29:00] **(A7)**
Basketball: Athens 2004 Olympics (Seven)	Women: AusBrazil_ 1,2,3-[19:50,19:41,4:20] **(B1)** Women: Russia-USA_3-[19:58] **(B2)** Men: Australia-USA_1,2-[29:51,6:15] **(B3)**
Basketball: Athens 2004 Olympics (SBS)	Men: USA-Angola_2,3-[22:25,15:01] **(B4)** Women: Australia-USA_1,2-[24:04-11:11] **(B5)**

Table 2. Sample Video Data.

4.1 Performance of Play-Break Segmentation

Play-break scoping plays a significant role to ensure that we can extract all of the features that usually exist in each highlight. Moreover, the statistics (especially play-/break-dominance) will be affected when the play-break sequences are detected perfectly. Table 3 to Table 5 depicts the performance of the play-break segmentation algorithm on soccer, AFL and basketball videos, respectively. It is to be noted that that *RC* = Replay-based (P-B sequence) Correction, *PD* = perfectly detected, *D* = detected, *M* = missed detection, *F* = false detection, *Tr* = Total number in Truth, *Det* = Total Detected, *RR* = Recall Rate, *PR*= Precision Rate, and *PD decr* = perfectly detected decrease rate if *RC* is not used; *Tru*= *PD*+*D*+*M*, *Det* = *PD*+*D*+*F*, *RR* = (*PD*+*D*+*M*)/*Tru* * 100%, *PR*= (*PD*+*D*)/*Det* * 100%, and *PD_Decr* = (PD-D)/PD * 100%. The results demonstrate that *RC* is generally useful to improve the play-break segmentation performance. It is due to the fact that many (if not most) replay scenes, especially soccer and AFL use global (i.e. play) shots. This is shown by all *PD_decr*, *RR*, and *PR* as *RC* always improves all of these performance statistics. In particular, the *RR* and *PR* for soccer 1-1 with *RC* are 100% each but they are reduced to below 50% without *RC*. In soccer 1-1 without *RC*, the *PD* dropped from 49 to 12 (i.e. 75% worse) whereas *M* increases from 0 to 25 and *F* increases from 0 to 5. This is due to the fact that soccer1 video contains many replay scenes which are played abruptly during a play, thereby causing a too-long play scene and missing a break. However, based on the statistics shown in Table 5, *RC* for basketball may not be as important as that of soccer and AFL. It is because basketball's replay scene uses more break shots such as zoom-in and close-up, as compared to soccer and basketball.

4.2 Performance of Soccer Events Detection

Based on Table 6 and Table 7, most soccer highlights can be distinguished from non-highlights with high recall and precision. As there are normally not many goal highlights in a soccer match, it would be ideal to have a high *RR* over a reasonable *PR*; 5 out of 7 goals are correctly detected from the 5 sample videos while 2 shoots and 1 non-highlight are classified as goals. The shoot segments detected as goals very exciting and nearly result in goal. On the other hand, the non-highlight detected as a goal also consist of a long duration and replay scenes and excited commentaries due to a fight between players. The foul detection is also effective as the *RR* is 81% and most of the misdetections are either detected as shoot or non which have the closest characteristics. However, the PR is considerably low since some shoots and non-highlights are detected as foul. An alternative solution is to use whistle existence for foul detection, but we still need to achieve a really accurate whistle detection that can overcome the high-level of noise in most of sport domains. Only 46 out of 266 non-highlight sequences were incorrectly detected as highlights. These additional highlights will still be presented to the viewers as there are generally not many significant events during a soccer video. In fact, most of these false highlights can still be interesting for some viewers as they often consist of long excitement, near-goal duration and replay scene.

4.3 Performance Basketball Events Detection

Highlights detection in basketball is slightly harder than soccer and AFL due to the fact that: 1) goals are generally not celebrated as much as soccer and AFL, 2) non-highlights are often detected as goal and vice versa. Fortunately, non-highlights mainly just include ball out play which hardly happen in basketball matches. Thus, we have decided to exclude non-highlight detection and replace it with timeout detection which can be regarded as non-highlights for most viewers. However, for some sport fans, timeouts may still be interesting to show the players and coaches for each team and some replay scenes. In addition to these problems, sequences containing fouls are sometimes inseparable from the resulting free throws. For such cases, the fouls are often detected as goal due to the high amount of excitement and long near-goal. However, fouls which are detected as goals can actually be avoided by applying a higher minimum highlight point for goal but at the expense of missing some goal segments. For our experiment, we did not use this option as we want to use a universal threshold for all highlights.

Based on Table 8 and Table 9, basketball goal detection achieves high *RR* and reasonable *PR*. This is due to the fact that goals generally have very unique characteristics as compared to foul and free throw. Timeouts can be detected very accurately (high RR and *PR*) due to their very long and many replay scenes. Moreover, most broadcasters will play some in-between advertisements when a timeout is longer than 2 minutes, thereby increasing the close-up ratio. Free throw is also detected very well due to the fact that free throw is mainly played in near-goal position; that is, the camera focuses on capturing the player with the ball to shoot. However, it is generally distinguishable from goal based on: less excitement, higher near goal, and more close-up shott; that is, goal scorer is often just shown with zoom-in views to keep the game flowing. However, the system only detected 28 out of 54 foul events. This problem is caused by the fact that after foul, basketball videos often abruptly switches to a replay scene which is followed by time-out or free-throw. This can be fixed with the introduction of additional knowledge such as whistle-detection.

4.4 Performance of AFL Events Detection

As shown in Table 10 and Table 11, the overall performance of the AFL highlights detection is found to yield promising results. All 37 goals from the 7 videos were correctly detected. Although the RR of behind detection seems to be low, most of the miss-detections are actually detected as goal. Moreover, behind is still a sub-type of goal except that it has lower point awarded. The slightly lower performance for detection of mark and tackle detection is caused by the fact that our system does not include whistle feature which is predominantly used during these events. Based on the experimental results, mark is the hardest to be detected and needs additional knowledge. In Table 11, *PR* and *RR* for behind is N/A as 1 behind was detected as goal while Mark = N/A because 5 marks were detected as goal.

5 Conclusion and Future Work

We have proposed an extensible approach for detecting events in sports video. The use of play-break scoping for all highlights have enabled us to obtain statistical-phenomena of the features contained in each highlight. Since the rules for highlight classification are driven by the statistics, none or low amount of domain-specific knowledge is required. Therefore, the proposed algorithms should be more robust for different sports, especially, field-ball goal oriented games. Based on the experimental results, play-break sequences are proven to be effective containers for detecting highlights. Thus, play-breaks need to be perfectly segmented and we have shown that replay-correction improves the performance. We have also proposed a segment-event based video data model which is designed using semi-schema-based and object-relationship modeling schemes. The schema is developed into XML schema with ORA-SS notation. The proposed schema is extensible as it supports incremental development of algorithms for feature-semantic extraction. Moreover, the schema does not need to be complete at one time while allowing users to add additional elements. We have also emphasized the usage of referencing relationship to avoid redundant data. Referencing also allows the system to add segments and events to achieve more straightforward and faster data insertions. In order to further verify and improve the robustness of the proposed algorithms for events detection we have incorporated more sport genre such as volleyball, tennis and gymnastics, into the existing dataset. The extracted information will allow the system to construct a larger sample of video database data which consequently would verify the benefits from using the proposed video indexing model.

Video	Soccer Play-break detection								
	PD	*D*	*M*	*F*	*Tru*	*Det*	*RR*	*PR*	*PD_decr*
S1-1 (*RC*)	49	0	0	0	49	49	100.00	100.00	
S1-1	12	12	25	5	49	54	48.98	44.44	75.51
S1-2(*RC*)	53	0	0	1	53	54	100.00	98.15	
S1-2	36	10	7	1	53	54	86.79	85.19	32.08
S2-1(*RC*)	54	1	1	12	56	68	98.21	80.88	
S2-1	53	2	1	12	56	68	98.21	80.88	1.85
S2-2(*RC*)	58	1	0	7	59	66	100.00	89.39	
S2-2	55	4	0	7	59	66	100.00	89.39	5.17
S3-1 (*RC*)	49	0	0	4	49	53	100.00	92.45	
S3-1	45	4	0	5	49	54	100.00	90.74	8.16
S3-2 (*RC*)	69	0	0	3	69	72	100.00	95.83	
S3-2	65	4	0	5	69	74	100.00	93.24	5.80
S4-1(*RC*)	49	0	0	9	49	58	100.00	84.48	
S4-1	40	8	1	13	49	62	97.96	77.42	18.37
S4-2(*RC*)	47	0	0	9	47	56	100.00	83.93	
S4-2	36	11	0	12	47	59	100.00	79.66	23.40
S5 (*RC*)	48	0	0	0	48	48	100.00	100.00	
S5	24	16	8	1	48	49	83.33	81.63	50.00

Table 3. Play-Break Detection in Soccer Videos.

	AFL Play-break detection								
Video	PD	D	M	F	Tru	Det	RR	PR	PD decr
A1 (*RC*)	34	0	0	5	34	39	100.00	87.18	
A1	29	5	0	8	34	42	100.00	80.95	14.71
A2 (*RC*)	21	6	0	8	27	35	100.00	77.14	
A2	16	10	1	5	27	32	96.30	81.25	23.81
A3 (*RC*)	20	3	0	4	23	27	100.00	85.19	
A3	17	6	0	6	23	29	100.00	79.31	15.00
A4 (*RC*)	29	0	0	1	29	30	100.00	96.67	
A4	21	6	2	2	29	31	93.10	87.10	27.59
A5 (*RC*)	34	0	0	1	34	35	100.00	97.14	
A5	23	4	7	3	34	37	79.41	72.97	32.35
A6 (*RC*)	50	2	0	3	52	55	100.00	94.55	
A6	36	10	6	7	52	59	88.46	77.97	28.00
A7 (*RC*)	41	10	4	4	55	59	92.73	86.44	
A7	39	12	4	6	55	61	92.73	83.61	4.88

Table 4. Play-Break Detection Results in AFL Videos.

Video	Basketball Play-break detection								
	PD	D	M	F	Tru	Det	RR	PR	PD decr
B1 (*RC*)	32	6	2	3	40	43	95.00	88.37	
B1	31	7	2	4	40	44	95.00	86.36	3.13
B2 (*RC*)	19	2	0	2	21	23	100.00	91.30	
B2	18	3	0	3	21	24	100.00	87.50	5.26
B3 (*RC*)	39	3	0	1	42	43	100.00	97.67	
B3	38	4	0	2	42	44	100.00	95.45	2.56
B4 (*RC*)	26	5	2	2	33	35	93.94	88.57	
B4	25	6	0	3	31	34	100.00	91.18	3.85
B5 (*RC*)	39	0	1	1	40	41	97.50	95.12	
B5	25	13	2	5	40	45	95.00	84.44	35.90

Table 5. Play-Break Detection Results in Basketball.

Ground truth	Highlight classification of 5 videos				
	Goal	Shoot	Foul	Non	Truth
Goal	**5**	0	2	0	7
Shoot	2	**66**	32	12	112
Foul	0	13	**91**	13	117
Non	1	11	34	**220**	266
Detected	8	90	159	245	

Table 6. Events Detection Results in Soccer Videos.

	S1		S2		S3		S4		S5		**Average**	
	RR	PR	RR	PR	RR	PR	RR	PR	RR	PR	**RR**	**PR**
Goal	60	100.0	100	50.0	N/A	N/A	100	33.3	N/A	N/A	**86.7**	**61.1**
Shoot	39.4	76.5	64.0	84.2	80.0	66.7	78.9	71.4	40.0	66.7	**60.5**	**73.1**
Foul	85.2	53.5	68.0	53.1	71.4	78.9	88.9	38.1	92.9	52.0	**81.3**	**55.1**
Non	86.5	82.1	86.3	88.5	90.5	90.5	75.8	100.0	60.0	80.0	**79.8**	**88.2**

Table 7. Distribution of Soccer Events Detection

Ground truth	Highlight classification of 5 basketball videos				
	Goal	Free throw	Foul	Timeout	Truth
Goal	**56**	0	0	2	58
Free throw	4	**14**	0	0	18
Foul	21	2	**28**	3	54
Timeout	0	0	0	**13**	13
Total Detected	81	16	28	18	

Table 8. Basketball Events Detection Results

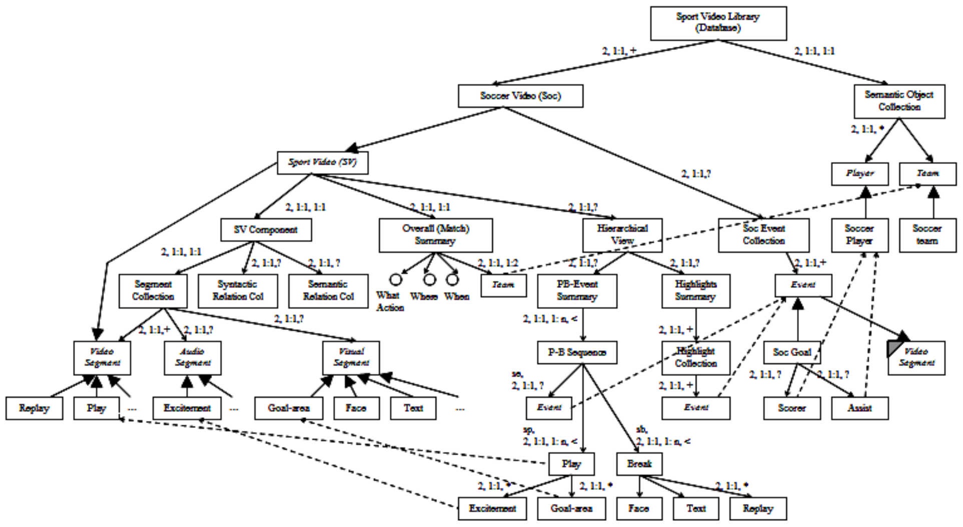


Figure 9. Extensible Indexing Scheme (3).

	B1		B2		B3		B4		B5		Average	
	RR	PR	RR	PR	RR	PR	RR	PR	RR	PR	RR	PR
Goal	100	72.2	75	50.0	95	70.4	100	72.2	100	66.7	**94**	**66.3**
Free throw	100.0	66.7	100.0	75.0	80.0	100.0	50.0	100.0	66.7	100.0	**79.33**	**88.3**
Foul	64.7	100.0	50.0	100.0	30.8	100.0	37.5	100.0	75.0	100.0	**51.59**	**100.0**
Timeout	100.0	100.0	100.0	50.0	100.0	40.0	100.0	66.7	100.0	100.0	**100**	**71.3**

Table 9. Distribution of Basketball Events Detection

Ground truth	Highlight classification of 7 videos					
	Goal	Behind	Mark	Tackle	Non	Truth
Goal	**37**	0	0	0	0	37
Behind	11	**12**	7	0	2	32
Mark	15	1	**35**	8	5	64
Tackle	4	0	9	**20**	**2**	35
Non	4	4	11	3	**33**	55
Detected	71	17	62	31	42	

Table 10. Events Detection Results in AFL Videos

	A1		A2		A3		A4		A5		A6		A7		AVG	
	RR	PR	RR	PR	RR	PR	RR	PR	RR	PR	RR	PR	RR	PR	RR	PR
Goal	100.0	44.4	100.0	52.9	100.0	57.1	100.0	33.3	100.0	50.0	100.0	63.6	100.0	53.3	100.0	50.7
Behind	50.0	100.0	N/A	N/A	33.3	33.3	33.3	66.7	50.0	100.0	33.3	100.0	33.3	50.0	38.9	75.0
Mark	50.0	60.0	N/A	N/A	60.0	60.0	77.8	77.8	60.0	42.9	66.7	42.1	47.1	80.0	60.3	60.5
Tackle	80.0	66.7	100.0	75.0	25.0	100.0	100.0	100.0	12.5	33.3	85.7	66.7	50.0	50.0	64.7	70.2
Non	77.8	100.0	33.3	100.0	50.0	50.0	50.0	66.7	71.4	71.4	46.2	100.0	75.0	69.2	57.7	79.6

Table 11. Distribution of AFL Events Detection

6 References

Adali, S., Candan, K. S., Chen, S.-S., Erol, K. and Subrahmanian, V. S. (1996) 'The Advanced Video Information System: Data Structures and Query Processing' *Multimedia Systems*, **4,** 172-186.

Connolly, T. M. and Begg., C. E. (2002) *Database systems : a practical approach to design, implementation, and management*, Addison-Wesley, Harlow [England] ; [New York].

Djeraba, C. (2002) 'Content-based multimedia indexing and retrieval' *Multimedia, IEEE*, **9,** 18-22.

Dobbie, G., Xiaoying, W., Ling, T. W. and Lee, M. L. (2000) In *Technical Report Department of Computer Science*,National University of Singapore.

Duan, L.-Y., Xu, M., Chua, T.-S., Qi, T. and Xu, C.-S. (2003) In *ACM MM2004*ACM, Berkeley, USA, pp. 33-44.

Ekin, A. and Tekalp, A. M. (2003a) In *International Conference on Mulmedia and Expo 2003 (ICME03)*, Vol. 1 IEEE, pp. 6-9 July 2003.

Ekin, A. and Tekalp, M. (2003b) 'Automatic Soccer Video Analysis and Summarization' *IEEE Transaction on Image Processing*, **12,** 796-807.

Han, M., Hua, W., Chen, T. and Gong, Y. (2003) In *Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, Vol. 2, pp. 950-954.

Li, B. and Ibrahim Sezan, M. (2001) In *Content-Based Access of Image and Video Libraries, 2001. (CBAIVL 2001). IEEE Workshop on*Practical, Sharp Labs. of America, Camas, WA, USA, pp. 132-138.

Nepal, S., Srinivasan, U. and Reynolds, G. (2001) In *ACM International Conference on Multimedia*ACM, Ottawa; Canada, pp. 261-269.

Oomoto, E. and Tanaka, K. (1997) In *The Handbook of Multimedia Information Management*(Ed, William I. Grosky, R. J. a. R. M.) Prentice Hall, Upper Saddle River, NJ, pp. 405 - 448.

Tjondronegoro, D., Chen, Y.-P. P. and Pham, B. (2004a) 'Integrating Highlights to Play-break Sequences for More Complete Sport Video Summarization' *IEEE Multimedia*, **Oct-Dec 2004,** 22-37.

Tjondronegoro, D., Chen, Y.-P. P. and Pham, B. (2004b) In *The 6th International ACM Multimedia Information Retrieval Workshop*ACM Press, New York, USA, pp. 267-274.

Wu, C., Ma, Y.-F., Zhang, H.-J. and Zhong, Y.-Z. (2002) In *Multimedia and Expo, 2002. Proceedings. 2002 IEEE International Conference on*, Vol. 1, pp. 805-808.

Xu, P., Xie, L. and Chang, S.-F. (1998) In *IEEE International Conference on Multimedia and Expo*IEEE, Tokyo, Japan,.

Using Formal Concept Analysis with an Incremental Knowledge Acquisition System for Web Document Management

Timothy J. Everts, Sung Sik Park and Byeong Ho Kang
School of Computing,
University of Tasmania
Sandy Bay, Tasmania 7001, Australia
{tjeverts, sspark, bhkang}@utas.edu.au

Abstract

It is necessary to provide a method to store Web information effectively so it can be utilised as a future knowledge resource. A commonly adopted approach is to classify the retrieved information based on its content. A technique that has been found to be suitable for this purpose is Multiple Classification Ripple-Down Rules (MCRDR). The MCRDR system constructs a classification knowledge base over time using an incremental learning process. This incremental method of acquiring classification knowledge suits the nature of Web information because it is constantly evolving and being updated. However, despite this advantage, the classification knowledge of the MCRDR system is not often utilised for browsing the classified information. This is because it does not directly organise the knowledge in a way that is suitable for browsing. As a result, often an alternate structure is utilised for browsing the information which is usually based on a user's abstract understanding of the information domain. This study investigated the feasibility of utilising the classification knowledge acquired through the use of the MCRDR system as a resource for browsing information retrieved from the WWW. A system was implemented that used the concept lattice-based browsing scheme of Formal Concept Analysis (FCA) to support the browsing of documents based on the MCRDR classification knowledge. The feasibility of utilising classification knowledge as a resource for browsing documents was evaluated statistically. This was achieved by comparing the concept lattice-based browsing approach to a standard one that utilises abstract knowledge of a domain as a resource for browsing the same documents.

Keywords: Formal Concept Analysis, Document Management, Knowledge Acquisition, Document Browsing

1 Introduction

The World Wide Web (WWW) has become the most popular information source for people today and is now the largest sharable and searchable repository of information (Park, Kim et al. 2003; Kim and Compton 2004). Originally the WWW widely utilised a passive information delivery mechanism that meant users would have to search for and then 'pull' down the information they needed. In order to overcome this limitation, a more active mechanism was required. This stemmed the research and development of software applications that could deliver the most up to date information in a timely manner. Web Monitoring Systems are an example of such software that has become popular in recent times (Liu, Pu et al. 2000; Tang, Liu et al. 2000; Boyapati, Chevrier et al. 2002; Liu, Tang et al. 2002). They check predefined target Web pages, automatically detect changes in these pages, and prompt users when these changes occur. The use of such systems appears to offer at least a partial solution to the problems of traditional information retrieval methods such as Web search engines, because the user has more control over the type and amount of information being delivered. It also ensures that the information being gathered is the latest.

However, the quantity of information being gathered can still be reasonably large. Subsequently, an effective method for storing and managing this information is also required. Document classification is one of the solutions to this problem. Traditionally, the dominant approach for classification is based on the content (text) of documents through trained classifiers using Machine Learning (ML) techniques because they achieve impressive levels of effectiveness (Sebastiani 2002). However, although classification by ML has proved to be successful in some commercial or research applications (Mladenic 1999), it is not generally appropriate for classifying information from the WWW. This is because the classification knowledge created during the training process cannot usually cater for the dynamic nature of Web documents. New information is constantly being generated or it is being updated. For this reason, efficient classification of documents retrieved from the WWW requires a technique that can operate on a continual learning process. This enables incremental knowledge acquisition that suits the dynamic nature of Web document information (Kim, Park et al. 2004).

A technique that has been found to be suitable for this purpose is the Multiple Classification Ripple-Down Rules

(MCRDR) knowledge acquisition method. Unlike machine learning methods, MCRDR constructs a classification knowledge base incrementally over time through a process of differentiation by the expert. When the case-based reasoning system of MCRDR retrieves cases that are recognised by the expert as inappropriate, the expert simply identifies the important characteristics of the present case that distinguish it from existing cases. In this way, knowledge is acquired by the system and new rules are created accordingly. When applied to Web Monitoring Systems, this technique enables the MCRDR rule set to be developed and adapted to suit the dynamic nature of Web documents (Park, Kim et al. 2003; Kim, Park et al. 2004).

Despite the appropriateness of using MCRDR to classify the documents collected by Web Monitoring Systems, the technique has one major weakness. MCRDR does not directly organise the knowledge in a way that is suitable for browsing (Kim and Compton 2004). As a result, the heuristic classification knowledge in an MCRDR knowledge base is not often utilised for browsing and searching the documents. Instead browsing and searching is facilitated through a structure based on some form of abstracted knowledge about the document domain that has been provided by the expert or user (Park, Kim et al. 2004). Therefore, it is suggested that the classification knowledge acquired through the use of MCRDR may also provide a useful resource for browsing the retrieved documents. To this extent, our research undertaken assessed the feasibility of utilising the heuristic classification knowledge of an MCRDR knowledge base as a resource for browsing documents in a specified domain. A system was developed and implemented that adopted the lattice-based browsing method of Formal Concept Analysis (Ganter, Stumme et al. 2005) as a means of providing a browsing representation based on heuristic classification knowledge. Formal Concept Analysis has been shown by Kim (Cole 2000; Kim and Compton 2001) to be quite successful for browsing documents in a specified domain. A comparative statistical analysis was performed between the use of a traditional browsing structure (based on abstract knowledge of a domain), and the concept lattice structure of FCA (based on heuristic classification knowledge). This has been done to evaluate the feasibility of utilising heuristic classification knowledge for browsing Web documents.

2 Related Work

2.1 WebMon and MCRDR

The WebMon Web Monitoring System was developed by a number of researchers at the University of Tasmania, Australia, and was built as part of the Personalised Web Information Management System detailed in Park et al. (Park, Kim et al. 2003). A Web monitoring system needs a method for archiving collected information effectively so it can be utilised in the future. For this purpose, WebMon adopts the MCRDR knowledge acquisition technique to classify and store retrieved documents appropriately.

The Multiple Classification Ripple Down Rules (MCRDR) method is derived from the Ripple Down Rules (RDR) method, a hybrid case-based and rule-based approach for knowledge acquisition and representation (Richards 2001). Knowledge acquisition (KA) in MCRDR involves the incremental addition of cases and justifications (rules) in the circumstance where a case is misclassified by the MCRDR system in the retrieval process. This incremental approach to KA is centred on the idea that the knowledge an expert provides is essentially a justification for a conclusion in a particular context (Compton and Jansen 1989; Preston et al. 1996). When the case-based reasoning (CBR) system of MCRDR retrieves a case(s) that is incorrect, the expert is required to identify the important characteristics that distinguish the incorrectly retrieved cases from the present case (Kang, Yoshida et al. 1997). It is thought that experts will select more valid knowledge if asked to deal with the differences between cases (Kang, Yoshida et al. 1997). Thus, the expert's justification provides a basis for a new rule to be created. The new rule(s) is first validated against existing rules (cornerstone cases) and then automatically appended to the knowledge base.

The MCRDR knowledge acquisition technique is used by the WebMon Web Monitoring System for determining where documents retrieved during Web monitoring should be stored for archival and sharing purposes. The structure used by the system to store the information is a storage folder structure (SFS). It is comparable to a hierarchical tree arrangement of folders, much like that used in common operating system environments such as Microsoft Windows. Depending on the choice of the user, the entire SFS can be defined up front or it can be defined incrementally as documents are collected. It is important to note that there are no predefined specifications that state the requirements for the specific folders contained in the SFS. The structure is usually devised based on the user's knowledge or understanding of the monitored document domain. It should also be noted that if the user chooses to utilise the Web portal option to share the collected information with other users, this same storage folder structure is replicated on the Web portal site. It is provided as a means for browsing and searching for the documents.

Once the SFS has been defined, newly updated Web documents retrieved during Web monitoring are classified into one or more target folders. Keywords are extracted from documents and form the conditions of rules in the MCRDR knowledge base. The rule conclusions are target folders in the SFS. This means that keywords in a newly retrieved document can be utilised in inference the MCRDR knowledge base, in order to recommend a target storage folder for the document. In the circumstance when a document is misclassified as a result of the inference process, the user simply adds knowledge to the knowledge base that enables a correct classification to be made.

As an example of the inference process for a document, Figure 1 shows how a document with the case (keywords) of [a,b,c,d,e,f,g] is recommended to storage locations within the SFS. The MCRDR KBS is drawn as an n-ary tree, with each node of the tree representing a rule which has a corresponding case. The inference process involves all rules attached to true parents being evaluated against the data. Thus the process begins by evaluating the root rule and then moving down level by level until either a leaf node is reached or none of the child nodes evaluate to true (Dazeley and Kang 2003). Since multiple pathways of refinement can be selected, multiple conclusions can be reached. This means that the last true rule on each pathway forms the conclusion for the case. Therefore, in the case presented in Figure 1, the inference process results in the recommendation of three storage folders for the current document, namely folders F_2, F_6, and F_5.

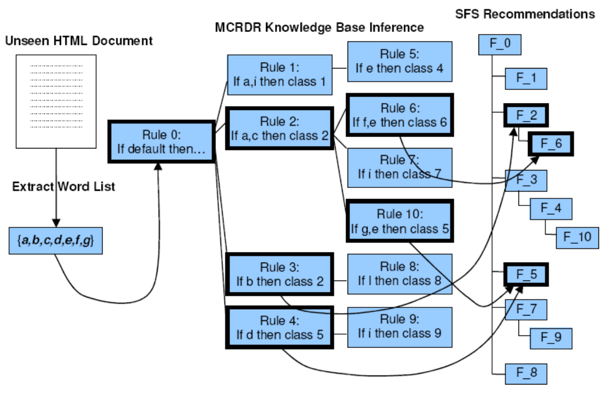

Unseen HTML Document
MCRDR Knowledge Base Inference
SFS Recommendations
Extract Word List
{a,b,c,d,e,f,g}
Rule 0: If default then…
Rule 1: If a,i then class 1
Rule 2: If a,c then class 2
Rule 3: If b then class 2
Rule 4: If d then class 5
Rule 5: If e then class 4
Rule 6: If f,e then class 6
Rule 7: If i then class 7
Rule 10: If g,e then class 5
Rule 8: If l then class 8
Rule 9: If i then class 9
F_0
F_1
F_2
F_6
F_3
F_4
F_10
F_5
F_7
F_9
F_8

Figure 1 - Inference for a Web Document Classification

Analysis of the WebMon Web Monitoring System reveals that the user (or domain expert) is utilising the devised SFS as a basis for defining a conclusion for document classifications. The common folder structure is used as a mediating knowledge representation for the user, and it enables them to easily build a conceptual document classification model using folder manipulation. In other words, the devised SFS is an explicit representation of the user's knowledge of the current document domain. Evidently, two types of knowledge are actually being utilised in the classification process. One type of knowledge is being used to define the SFS, while another type of knowledge is being used in the actual classification of documents to target folders. This point is more apparent when the user devises the SFS. Its structure is based upon their conceptual hierarchical understanding of the domain. However, when the user classifies a document to a folder in the storage structure, that classification is made based on the actual content of the document, namely keywords. These keywords may also be embedded in the conditions of the existing classification rules in the MCRDR knowledge base. The knowledge used in the creation of the SFS is hereafter referred to as being 'abstract domain knowledge'. In regards to the second type of knowledge, it is hereafter referred to as being 'heuristic classification knowledge', since it is associated with the classification knowledge embedded in the rules of the MCRDR knowledge base. Having discovered that there are two types of knowledge being utilised by WebMon for document classification, it is well worth noting that only the abstract domain knowledge is ever utilised for browsing the documents.

Although there are two potentially useful knowledge types which could be used as a basis for browsing documents, only one of them is currently being utilised by the majority of Web portal sites. This means WWW users are being forced into searching for documents using a user-defined structure which is based on abstract domain knowledge rather than on heuristic classification knowledge. It can be argued that the heuristic classification knowledge would be more appropriate for being used as a basis for browsing the documents, because it more accurately represents the actual content of each document. For this reason, the main suggestion of this research was that if the classification knowledge can be incorporated as the basis for a document browsing structure, it may also provide an extremely useful resource for browsing the documents in the domain. Therefore, it was proposed that the use of an alternate browsing method instead of the storage folder structure may enable classification knowledge to be utilised as a basis for browsing the documents classified by MCRDR. The approach suggested and adopted in this research was the lattice-based browsing scheme of Formal Concept Analysis, so therefore it is outlined in the section that follows.

2.2 Formal Concept Analysis

Formal Concept Analysis (FCA) is a mathematical approach used for conceptual data analysis and knowledge processing. It has had numerous applications for data analysis and information retrieval in fields such as medicine, psychology, ecology, social science and political science. Various researchers have shown that a quite successful method for browsing documents in a specified domain is the lattice-based browsing approach of Formal Concept Analysis (Cole 2000; Cole, Eklund et al. 2004; Kim and Compton 2004; Becker 2005; Carpineto and Romano 2005; Eklund and Wormuth 2005; Quan, Hui et al. 2005).

FCA 'formulates concepts in terms of objects and their properties or attributes, and provides a way of combining and organising individual concepts (of a given context) into [a] hierarchically ordered conceptual structure [known as a] … concept lattice structure' (Rajapakse and Denham 2003). Correia et al. (Correia, Willie et al. 2003) comments that concepts are necessary for expressing human knowledge and a formalisation of concepts acts as means of communicatively representing knowledge.

FCA is based on a formal understanding of a concept as a unit of thought, comprising its extension and intension. The extension (extent) of a formal concept is formed by all objects to which the concept applies (a set of objects) and the intension (intent) consists of all attributes existing in those objects (a set of attributes). The set of objects, set of attributes and the relations between an object and an attribute in a data set form the basic conceptual structure of FCA (known as a formal context). A formal context is defined as a triple (G, M, I) where I maps the relation between a set of objects G, and a set of attributes M. This is denoted formally as:

$$C = (G, M, I)$$

where C represents the context. In order to express that a particular object g is in a relation I with a particular attribute m, the relation is given by:

$$(g, m) \in I \text{ or } gIm$$

and should be read as "the object g has the attribute m".

Once a formal context has been defined, all the formal concepts of the formal context can be derived. A formal concept is represented as a pair (A, B), where A is a subset of objects of the formal context and B is a subset of attributes of the formal context. In order for a pair (A, B) to be a formal concept, all attributes common to objects in A, the intent, and all objects common to attributes in B, the extent, must be the same.

This duality relationship is formalised by:

1. Set of attributes common to the objects in A (intent)

$$A' = \{ m \in M \mid (g,m) \in I \text{ for all } g \in A \}$$

2. Set of objects common to the attributes in B (extent)

$$B' = \{ g \in G \mid (g,m) \in I \text{ for all } m \in B \}$$

The formal concepts of a formal context can be ordered and arranged hierarchically into a conceptual structure of FCA called a concept lattice. Ganter and Wille (1997) comment that concept lattices are useful for unfolding given data, ‘making their conceptual structure visible and accessible, in order to find patterns, regularities, exceptions etc.’ Therefore, the concept lattice structure provides a means of revealing the implicit relationships between data that are not otherwise obvious. The concept lattice is ordered by the smallest set of attributes (intent) between the concepts and thus maps an ordering from the most general to the most specific concept, top to bottom (Kim 2003).

To form the concept lattice, hierarchical subconcept - superconcept relations between all the formal concepts need to be found. This is formalised by $(A_1, B_1) \leq (A2, B2) :\Leftrightarrow A1 \subseteq A2 (\Leftrightarrow B2 \subseteq B1)$ where (A1, B1) is called a subconcept of (A2, B2), and (A2, B2) is called a superconcept of (A1, B1). ‘The relation $\leq$ is called the hierarchical order of the concepts’ (Kim 2003, p. 55). When the lattice is formed, the largest *subconcept* will be the top most concept in the lattice, called the *supremum*, and the smallest *subconcept* will be the bottom most concept, called the *infimum*.

2.3 Combining MCRDR with FCA for Browsing Documents

Various studies have shown that the lattice-based method of FCA can be utilised as an effective means for browsing documents in specialised domains. Kim (2003) developed a Document Management and Retrieval System (DMRS) for specialised domains on the WWW that utilised an incrementally built concept lattice as a means of browsing and retrieving documents. As part of her work, a user evaluation was performed on the browsing and retrieving of documents using the lattice structure. The evaluation concluded that users considered searching a specialised domain using lattice-based browsing to be more helpful than using Boolean queries and hierarchical browsing. Furthermore, users also found that the ad hoc evolvement of the lattice-based browsing structure provided good efficiency in retrieval performance. The lattice-based browsing approach has also been shown to be much more advantageous than a hierarchical approach to browsing documents, such as the storage folder structure used by WebMon (Kim and Compton 2004). In regards to utilising MCRDR classification knowledge in the lattice structure, research undertaken by Richards (Richards 1998) revealed that the rules of an RDR knowledge base can be utilised to generate an FCA concept lattice structure. Therefore, the lattice-based browsing method of FCA may be used as a means for defining an effective document browsing structure that is based on MCRDR heuristic classification knowledge. The feasibility of this could be tested by utilising the structure to browse the documents collected by the WebMon Web Monitoring system and comparing this to browsing the same documents using the system’s storage folder structure.

3 System Implantation

3.1 System Overview

In order to utilise the MCRDR heuristic classification knowledge as a basis for browsing the documents collected during the Web monitoring project, it was necessary to develop a system that implemented an alternate browsing representation. Subsequently, a system, called iWeb FCA, was developed as part of this research which utilised the MCRDR heuristic classification knowledge to generate a FCA concept lattice for browsing the documents. The iWeb FCA system generates a FCA concept lattice based on the MCRDR heuristic classification knowledge to provide an alternate browsing structure for the documents collected and classified by WebMon. In addition, the system is also capable of utilising the abstract domain knowledge embedded in the storage folder structure as a resource for generating a concept lattice. The system can be configured to generate a concept lattice using either one of the knowledge sources as a resource or it can be configured to utilise both resources at once for lattice generation.

In using the system to generate a concept lattice, it is important to note that documents are considered to constitute the objects used in FCA and the rule keywords (classification knowledge) or folder names (abstract domain knowledge) are considered to constitute the attributes. However, this approach does not strictly comply with the original formulation of FCA in which an object was implicitly assumed to have some sort of unity or identity so that the attributes applied to the whole object (e.g. a car has four wheels). As Kim (2003) states, ‘clearly documents do not have the sort of unity where attributes will necessarily apply to the whole document’. However, in order to use FCA in the iWeb FCA system, the following assumptions are made. Documents correspond to objects and the rule condition keywords used to classify a document or the names of the folders in which the document is stored constitute the attribute set. A similar approach has been shown by Kim (2003) to be quite feasible.

3.2 System Functionality

3.2.1 Reducing the Amount of Documents in the Domain

In order to evaluate the feasibility of utilising heuristic classification knowledge for browsing documents using an FCA lattice structure, it was only necessary to generate a single complete lattice for any formal context and gather statistical results about that generated lattice structure. However, the lack of available system resources and the significant quantity of documents for a single domain posed a problem for lattice generation. It was too time consuming to generate a complete concept lattice using the full set of documents. For this reason, iWeb FCA included a function that reduced the number of documents stored in all folders in the storage folder structure to contain, at a maximum, a specified amount. At a minimum, a folder could contain zero documents. Note the fact that the actual number of *folders* is not reduced means that all of the heuristic classification knowledge is still utilised to generate the complete concept lattice. This is because the MCRDR rules apply to particular folders in the storage folder structure, and not particular documents. In other words, the conclusions of the MCRDR rules are folders.

3.2.2 Generating a Complete Lattice

The batch process utilised to build the formal concepts and the concept lattice is an implementation of the general methodology of FCA for formulating concepts and building the concept lattice. The algorithm used in iWeb FCA was based upon the explanations of FCA provided by Richards (1998), Kim and Compton (2000), and Kim (2003). In detailing the procedure, C represents the formal context stored in iWeb FCA's database, D represents the set of objects (documents) in C, and M represents the set of attributes (rule keywords or folder names) in C. The procedure implemented is detailed in Figure 2.

Step 1:

Formulate an extent containing the set of objects G representing the largest concept of C. Then perform step 2 for each attribute m in the set M.

Step 2:

a) Find the set of objects X that contains the attribute m.

b) Check whether any previously formulated extent is equivalent to X.

c) If an equivalent extent of X does not exist, then add the set X as an extent of the attribute m.

d) Determine the intersection of X with all extents calculated in previous steps. If the intersection set does not exist, then add the intersection set as an extent of attribute m.

Step 3:

For each formulated extent, determine its intent:

$$Y \leftarrow \{ m \quad M \mid (g,m) \quad I \text{ for all } g \quad X\}$$

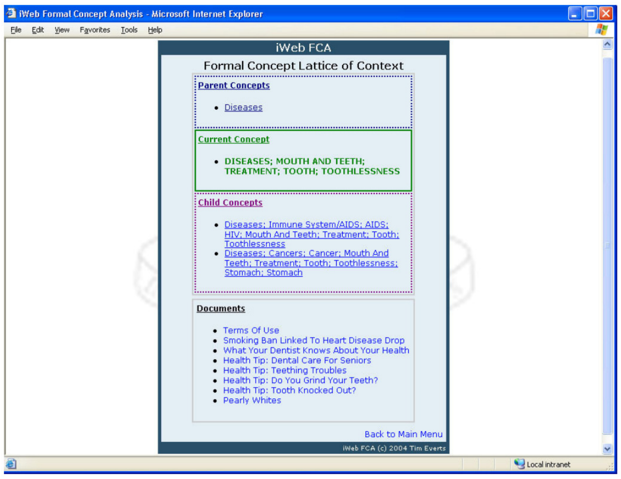
Step 4:

Construct the concept lattice by finding all the hierarchical ***subconcept*** *-* ***superconcept*** *relations between all the formal concepts of C that were computed in steps 1 to 3.*

Figure 2 – Procedure for Generation a Concept Lattice in iWeb FCA

3.2.3 Browsing the Concept Lattice

A sample of the concept lattice browsing interface used in iWeb FCA is shown in Figure 3. As in the approach of Kim and Compton (2000), the lattice display is simplified by showing only direct neighbour nodes of the current node using hyperlinks. Each lattice node represents a concept comprised of a pair (X,Y), where X is the extent (a set of documents) and Y is the intent (a set of classification rule keywords) of the concept. The intents of each concept are used for indexing the terms of the browsing structure.


iWeb Formal Concept Analysis - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Local intranet

iWeb FCA

Formal Concept Lattice of Context

Parent Concepts

- Diseases

Current Concept

- DISEASES; MOUTH AND TEETH; TREATMENT; TOOTH; TOOTHLESSNESS

Child Concepts

- Diseases; Immune System/AIDS; AIDS; HIV; Mouth And Teeth; Treatment; Tooth; Toothlessness
- Diseases; Cancers; Cancer; Mouth And Teeth; Treatment; Tooth; Toothlessness; Stomach; Stomach

Documents

- Terms Of Use
- Smoking Ban Linked To Heart Disease Drop
- What Your Dentist Knows About Your Health
- Health Tip: Dental Care For Seniors
- Health Tip: Teething Troubles
- Health Tip: Do You Grind Your Teeth?
- Health Tip: Tooth Knocked Out?
- Pearly Whites

Back to Main Menu

iWeb FCA (c) 2004 Tim Everts

Figure 3 – iWeb FCA Concept Lattice Browsing Interface

The concept lattice browsing interface in iWeb FCA is divided into four distinctly recognisable sections. The current lattice node is displayed in green in a section labelled 'Current Concept', while parent nodes and child nodes are listed as hypertext links in sections labelled 'Parent Concepts' and 'Child Concepts' respectively. The set of documents associated with the current node are listed as hypertext links in a section labelled 'Documents'. The actual browsing of the lattice begins from the root node (concept) and the relationships of concepts can be explored by traversing from vertex to vertex by clicking on a child or parent node hypertext link. Each time a new node is selected, the interface is updated to show the parent and child nodes of the current node. The list of documents associated with the current node is also refreshed. Documents at a node can be viewed by clicking the appropriate hypertext link and the document will be displayed in a new Web browser window.

4 Evaluation

4.1 Data Set

The MCRDR heuristic classification knowledge utilised in this research study was collected over a period of time during a project undertaken at the University of Tasmania in Hobart, Australia. Table 1 summarises the data created as a result of the Web monitoring project which was focused on the domain of e-Health. In total, 7 sites were monitored by WebMon and 7588 documents were retrieved from those sites. Of those 7588 documents, 4598 were classified to the storage folder structure which contained 119 folders. During the classification process, 172 rules were created and a total of 285 unique rule conditions (keywords) were contained in those rules. The iWeb Web Portal site divided the complete storage folder structure into various sub-domains of eHealth, based on the individual folders at the second level of the storage folder structure. These sub-domains included ‘Diseases’, ‘Demographic Groups’, ‘Drug Information’ and ‘Health and Wellness’. Dividing the complete storage folder structure into smaller parts simplified browsing for information, especially since the entire storage folder structure was quite large and the quantity of information was significant.

Web Monitoring	
Total Sites Monitored	7
Total Articles Collected	7588
Total Articles Classified	4598
Classification Knowledge	
Total Rule Used	172
Total Rule Conditions	285
Storage Folder Structure	
Total Folders	119

Table 1 – Summary of Web Monitoring Project

To conduct evaluation, a sub-domain of the eHealth domain was first selected to be utilised as the source of data for generating the concept lattice. The reason why only a sub-domain was selected is because the limited system resources available meant it would take a significant amount of time to generate a single complete concept lattice for the entire eHealth domain. Also, since the storage folder structure could be distinctly divided into the various sub-domains of eHealth (as is done on the iWeb Web portal site), it was much simpler to just deal with a small portion of the overall structure for the purpose of analysing it. Consequently, the sub-domain of ‘Diseases’ was selected for the purpose of the analysis. It contained the most information out of all the sub-domains and also had the largest storage folder structure. To enable a concept lattice to be generated from the Diseases sub-domain data, iWeb FCA was used to reduce the number of documents in any folder to be no more than 32. This figure was chosen through a trial and error approach based on the amount of time it took to generate a concept lattice with the available system resources. It resulted in a total number of 1063 classified documents making up the reduced data set.

4.2 Method

Having reduced the source domain data to a manageable amount for lattice generation, iWeb FCA was used to generate two different types of concept lattices. The first concept lattice was generated based on the MCRDR heuristic classification knowledge, and the second concept lattice was generated based on a combination of MCRDR heuristic classification knowledge (rule keywords) and abstract domain knowledge (folder names) because many of the folder names used in abstract domain knowledge also occur as keywords in the heuristic classification knowledge. For this reason, it may also be potentially useful to browse documents using a combination of the two knowledge types, especially because often a user does not make a clear distinction between the two knowledge types. Therefore, browsing a concept lattice based on this combination of knowledge types was also assessed as part of the evaluation undertaken.

The final step of the evaluation procedure was to gather and record statistics on the different browsing structures. This was done in order to assess the feasibility of utilising heuristic classification knowledge for browsing documents. Three main forms of analysis were performed. Firstly, the physical composition of the different browsing structures was analysed as a means of assessing the implications that each would have on browsing for documents. Secondly, the distribution of documents in the browsing structures was compared to determine whether utilising heuristic classification knowledge as a resource for browsing enhances a user’s ability to locate a particular document. Finally, an analysis was performed on how the structures would actually be browsed. This final analysis was achieved by programmatically simulating the browsing process and recording information about each level that would be traversed in each browsing structure. The results and discussion of the analytical evaluation are presented in the Section that follows.

5 Result

5.1 Physical Browsing Structures

Table 2 shows the main statistics gathered from analysing the physical composition of the storage folder structure (SFS). Table 3 shows the statistics gathered from analysing the physical composition of a concept lattice which was generated based on the MCRDR heuristic classification knowledge (HCK lattice), as well as statistics for a second concept lattice generated on a combination of MCRDR heuristic classification knowledge and abstract domain knowledge (HCK-ADK lattice).

Total Number of Folder	80
Folders with Documents	56
Folders without Documents	24
Average Sub-Folders per Folder (without leaf folders)	6.08
Total Rules Utilised	78
Total Rule Keywords	109

Table 2 – Summary of Storage Folder Structure

	HCK	HCK-ADK
Total Number of Nodes (Concept)	77	88
Total Nodes with Documents	76	87
Total Nodes without Documents	1	1
Number of Single Level Nodes	22	3
Average Child Nodes per Node	1.69	1.69
Average Attributes per Node	4.08	7.18

Table 3 – Summary of Concept Lattice Structure

By comparing the physical composition of the SFS (see Table 2) with the HCK and HCK-ADK concept lattice structure (see Table 3), the implications of browsing documents based on heuristic classification knowledge as opposed to abstract domain knowledge can be made clear. In the SFS there is an average of 6.08 sub-folders for every folder (excluding leaf folders), while in the HCK and HCK-ADK lattice there is an average of 1.69 children nodes per node. Since the SFS is a hierarchical tree structure, it would be traversed starting from the root folder and finishing at a leaf folder. This means that in browsing the SFS a user tries to pick the best sub-folder at each step in order to locate a particular document. Each time a document is not located in a particular folder, the user would have to make the decision between an average of about 6 sub-folders as to where to go next. This also means that if a leaf folder is reached, it is difficult to know what to do next because the best guesses have already been made at each decision point.

However, with the HCK and HCK-ADK lattice structure, making the decision of where to go next is much less overwhelming for the user. This is because on average there is only about 1 or 2 child nodes to choose from. Also, since the HCK and HCK-ADK lattice is more of a network type structure, it means that if a document is not located by taking one path, it is possible to go back up another path rather than starting again. This opens up new decisions which have not previously been considered.

A further interesting aspect of utilising the HCK and HCK-ADK lattice for browsing documents is that every node except one (which would be the bottom-most node) contains at least one document (see Table 3). However, in the SFS there are 24 folders that do not contain any documents (see Table 2). This means there are 24 possible decisions a user could make when browsing the SFS that are potentially useless in locating a particular document. This not only makes locating a document more difficult in the SFS, but it would no doubt also increase a user's frustration.

Comparing the physical structure of the HCK-ADK lattice with the structure of the HCK lattice (Table 3) produces some very interesting results. The most interesting result is the significant decrease in the amount of single level nodes in the HCK-ADK lattice. In this analysis, a single level node is a node that has the *supremum* node (top most concept in the lattice) as its only predecessor, and the *infimum* node (bottom most concept in the lattice) as its only successor. If a large percentage of the total nodes in a lattice are single level nodes, it implies that the overall lattice structure is very shallow, meaning that more of the concepts will be general in nature. In regards to browsing the lattice for documents, this implies it will be more difficult for a user to locate the document desired. This is because there are fewer concepts in the lattice that would be specific enough to uniquely represent the attributes of that document.

Calculating the percentage of single level nodes in each lattice generated reveals that even though the HCK-ADK lattice contains 10 extra nodes (88 nodes) than the HCK lattice (77 nodes), only about 3 percent of nodes in the HCK-ADK lattice are single level nodes. However, in the HCK lattice, about 29 percent of all nodes are single level nodes. This implies that it would be much easier to locate a particular document when browsing the HCK-ADK lattice because a larger number of terms are being used to represent the attributes of documents resulting in a greater number of more specific concepts being generated.

5.2 Distribution of Documents

A second statistical analysis was undertaken to analyse how documents were distributed in the various browsing structures. The aim of this analysis was to determine whether utilising heuristic classification knowledge as a resource for browsing enhances a user's ability to locate a particular document.

The most significant result from analysing the distribution of documents in the SFS shows that the majority of the total 1063 classified documents are only located in a single folder. This implies that it would be quite difficult to locate a particular document when browsing the SFS because few documents can be found in multiple folders. Consequently, this makes the decision of which folders a user selects in searching for a document a lot more critical, since the likelihood of finding the document in a particular folder is relatively small.

The ability to locate a document can be significantly improved if the heuristic classification knowledge and abstract domain knowledge are used as a resource for browsing instead. In the HCK and HCK-ADK lattice, documents are distributed much more evenly than in the SFS. As a result, a larger amount of documents are located at a higher number of multiple locations (nodes) in the HCK and HCK-ADK lattice. This is also evident when the distribution of documents between the SFS, HCK and HCK-ADK lattice are compared graphically, as shown in Figure 4.

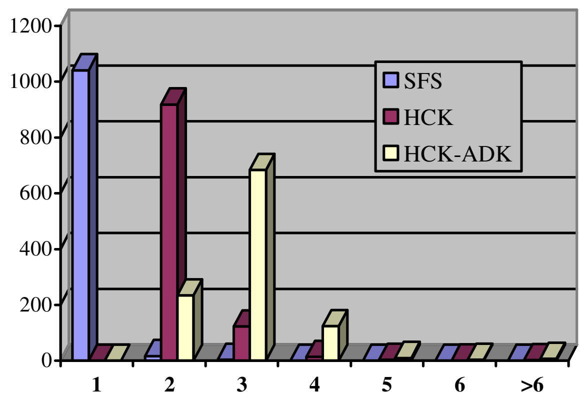


Figure 4 – Distribution of Documents in Multiple Locations

It is interesting to note the effect that utilising the terms from both knowledge types has on the distribution of documents in the lattice structures. In the HCK-ADK lattice, the distribution of documents appears to be more evenly spread than in the HCK lattice. This can be clearly seen in Figure 4. Also, in the HCK-ADK lattice, 78 percent of documents are located at 3 or more nodes, whereas only about 14 percent are located at that many nodes in the HCK lattice. This shows that the utilisation of the terms of both knowledge types can also provide more possibilities for locating a document while browsing.

5.3 Analysis of Browsing

The final statistical analysis undertaken involved simulating the way a user might actually browse each of the different structures. For the storage folder structure (SFS) this was simulated programmatically by beginning at the first level of browsing, namely the root folder and recording information about the properties of that browsing level. Then the entire SFS was traversed one level (folder) deeper to all sub-folders visible from the first level, and the properties of that level were also recorded. This process continued until it was not possible to traverse any deeper, namely when all folders on the browsing level were leaf folders.

A similar programmatic simulation was also applied to the generated concept lattices to record the information about each level of browsing in the lattice structure. The deepest level of browsing in the lattice was the level that contained only the *infimum* node (bottom most concept in the lattice). It should be noted that the structure of a concept lattice is such, that when browsing the lattice an individual node may appear (be visible) at two different browsing depths, depending on which path is taken through the lattice.

The statistics that were recorded at each level of browsing included the total number of folders or nodes for that level, the total number of documents, the total number of unique documents, and the average number of documents per folder or node on that level.

(a) Storage Folder Structure

Browsing Depth (folders)	Total Folders	Total Docs	Unique Docs	Average Docs per Folder
1 Level	1	0	0	0.00
2 Level	20	489	487	24.45
3 Level	59	602	586	10.20

(b) HCK Concept Lattice

Browsing Depth (Nodes)	Total Nodes	Total Docs	Unique Docs	Average Docs per Node
1 Level	1	1063	1063	1063.0
2 Level	46	1088	1063	23.65
3 Level	25	152	145	6.08
4 Level	6	9	7	1.50
5 Level	1	2	2	2.0
6 Level	1	0	0	0.00

(c) HCK-ADK Concept Lattice

Browsing Depth (Nodes)	Total Nodes	Total Docs	Unique Docs	Average Docs per Node
1 Level	1	1063	1063	1063.00
2 Level	21	1166	1063	55.52
3 Level	46	852	829	18.52
4 Level	20	68	60	3.40
5 Level	5	8	6	1.60
6 Level	1	2	2	2.00
7 Level	1	0	0	0.00

Table 4 – Analysis of Browsing

Table 4 presents the statistics gathered by simulating browsing the storage folder structure (SFS), the HCK lattice and the HCK-ADK lattice.

The first and perhaps most obvious comparison that can be made between the SFS and HCK and HCK-ADK lattice is the difference in the number of browsing levels. Starting at the root folder (level 1) in the SFS, it is possible to traverse to a maximum browsing depth of 3 levels. On the other hand, in the HCK lattice it is possible to traverse to a maximum browsing depth of 6 levels and in the HCK-ADK lattice to 7 levels.

The SFS appears to be much easier for a user to browse because there are fewer levels of browsing in it. However, the fact that there are fewer levels of browsing means that the amount of folders on each level is quite large. The structure of the SFS is such, that the deeper the user browses, the larger the amount of folders that appear on each level. This means the decision of which folder to select when trying to locate a document becomes much more difficult with each new level that is traversed. In the HCK lattice the opposite is the case. Disregarding the first level of browsing (the root node), the deeper a user browses the HCK lattice structure, the fewer the nodes that appear at each browsing level. Therefore the decision of where to go next when browsing the HCK lattice only becomes easier rather than more difficult.

It is also interesting to compare the total number of documents and unique documents at each level of browsing in the SFS and the HCK / HCK-ADK lattice (see Table 4). Since the SFS only has three levels, it is appropriate to compare only the first three levels of both structures. This comparison reveals that all 1063 classified documents can be located at both of the first two levels of browsing in the HCK / HCK-ADK lattice, while not even half of all the documents can be found at each of the same two levels of browsing in the SFS. This would suggest that there is more chance of locating a desired document in the HCK / HCK-ADK lattice as there is in the SFS.

Comparing the difference between the HCK-ADK lattice and the HCK lattice shows that there is only one extra level of browsing in the HCK-ADK lattice. Another interesting statistic is that the average number of documents per node on nearly all the levels of browsing in the HCK-ADK lattice is significantly higher than that in the HCK lattice. Furthermore, the overall difference between the number of total and unique documents on each level in the HDK-ADK lattice is also significantly higher than in the

HCK lattice. Therefore, from the comparisons presented it can be concluded that the utilisation of the terms of both knowledge types improves the possibility of locating a document during browsing. This makes the browsing experience all the more beneficial for a user.

6 Conclusion

The investigation undertaken in this study was aimed at determining the feasibility of utilising heuristic classification knowledge acquired through the use of MCRDR as a resource for browsing documents retrieved from the WWW. A Web-based system was developed which generated a FCA concept lattice using the heuristic classification knowledge of MCRDR. To evaluate the feasibility of utilising heuristic classification knowledge as a resource for browsing documents, a comparative statistical analysis was performed. This involved comparing the difference between browsing documents using two different structures. Namely, a storage folder structure (SFS) based on abstract knowledge of a domain, and a concept lattice based on MCRDR heuristic classification knowledge.

From the evaluation performed, it is concluded that the concept lattice-based browsing scheme of FCA provides a feasible way to utilise MCRDR heuristic classification knowledge for browsing documents of a specific domain. An analysis of the physical composition of the SFS compared with the concept lattice structure revealed that browsing based on heuristic classification knowledge significantly simplifies each decision a user has to make during browsing. Also, analysing the distribution of documents in each browsing structure revealed that a user's ability to locate a particular document when browsing the lattice structure is significantly enhanced. Documents are more evenly distributed throughout the lattice than in the SFS, and they can also be found in a larger number of multiple locations. Furthermore, by programmatically simulating the way a user might browse each structure, it was possible to determine the options they would be presented with during browsing. Even though the lattice structure based on heuristic classification knowledge appeared to require more interaction from a user during browsing than when using the SFS, the browsing experience is much less overwhelming because each individual stage of browsing is much simpler.

In addition, the results of a secondary investigation concluded that using the terms of both abstract domain knowledge and heuristic classification knowledge also presents itself as a viable option for browsing documents. Statistically comparing a lattice generated on the terms of both knowledge types with a lattice generated plainly on heuristic classification knowledge produced some interesting results. The results showed that the utilisation of the terms of both knowledge types provides a much richer context for browsing. Each document can not only be found at a larger number of multiple locations in the lattice, but the extra terms also enable the location of each document to be identified more specifically.

7 Further Work

There are potentially several areas of research related to this study that can be investigated. An immediate continuation of the work undertaken might be to incorporate the prototyped concept lattice browsing approach of iWeb FCA into the iWeb Web Portal Site. This may be useful for providing an alternate method to users for browsing documents on that site, especially considering the significant quantity of information available.

An aspect that was not covered by this study is a user's actual satisfaction of browsing documents based on heuristic classification knowledge, as compared with browsing based on abstract domain knowledge. To evaluate this would also be interesting and would most likely involve performing a quantitative user study. The study could compare and assess the performance of browsing documents based on each type of knowledge.

It may also be interesting to investigate the use of other classification knowledge types as a resource for browsing documents. This study simply utilised the classification knowledge of MCRDR because it was readily available and suitable. There may well be other types of classification knowledge that can be utilised appropriately for browsing documents. In the same manner, it may also be useful to evaluate the use of an alternate browsing structure, other than the concept lattice of FCA, that can also utilise heuristic classification knowledge as a resource for browsing documents.

However, perhaps the most interesting point that remains to be seen is whether browsing schemes based on heuristic classification knowledge will become a standard for browsing information on the WWW. With the consistent increase in the amount of information being generated on the WWW, there is an increasing need for more effective and simple ways of locating and retrieving information. To this extent, the utilisation of heuristic classification knowledge as a resource for browsing and searching of information may provide a potential solution to this problem.

8 References

Becker, P. (2005). "Using intermediate representation systems to interact with concept lattices." Formal Concept Analysis. Third International Conference, ICFCA 2005. Proceedings (Lecture Notes in Artificial Intelligence Vol.3403): 265-268.

Boyapati, V., K. Chevrier, et al. (2002). ChangeDetector[tm]: a site-level monitoring tool for the WWW. WWW 2002.

Carpineto, C. and G. Romano (2005). Formal Concept Analysis as Mathematical Theory of Concepts and Concept Hierarchies. Formal Concept Analysis Foundations and Applications. B. Ganter, G. Stumme and R. Wille.

Cole, R. J. (2000). The Management and Visualisation of Document Collections Using Formal Concept Analysis, Griffith University: 122.

Cole, R. J., P. W. Eklund, et al. (2004). Browsing Semi-Structured Texts on the Web Using Formal Concept Analysis. Intelligent Technologies for Information Analysis. N. Zong and J. Liu, Springer: 243-264.

Correia, J. H., G. Willie, et al. (2003). "Concpetual Knowledge Discovery - A Human-Centered approach." Applied Artificial Intelligence **17**: 281-302.

Dazeley, R. and B. Kang (2003). Weighted MCRDR:Deriving Information about Relationships between Classifications in MCRDR. 16th Australian Joint Conference on Artificial Intelligence (AI'03), Perth, Australia.

Eklund, P. and B. Wormuth (2005). "Restructuring help systems using formal concept analysis." Formal Concept Analysis. Third International Conference, ICFCA 2005. Proceedings (Lecture Notes in Artificial Intelligence Vol.3403): 129-144.

Ganter, B., G. Stumme, et al. (2005). Formal Concept Analysis Foundations and Applications.

Garter, B. and R. Willie (1997). "Applied Lattice Theory:Formal Concept Analysis."

Kang, B., K. Yoshida, et al. (1997). "Help desk system with intelligent interface." Applied Artificial Intelligence **11**(7-8): 611-631.

Kim, M. (2003). Document Management and Retrieval for Specialised Domains: An Evolutionary User-Based Approach, University of New South Wales.

Kim, M. and P. Compton (2000). Developing a domain-specific Information Retrieval Mechanism. 6th Pacific Knowledge Acquisition Workshop (PKAW 2000), Sydney Australia.

Kim, M. and P. Compton (2001). Formal Concept Analysis for Domain-Specific Document Retrieval Systems,. 13th Australian Joint Conference on Artificial Intelligence (AI'01), Adelaide Australia, Springer-Verlag.

Kim, M. and P. Compton (2004). "Evolutionary document management and retrieval for specialized domains on the web." International Journal of Human-Computer Studies **60**(2): 201 - 241.

Kim, Y. S., S. S. Park, et al. (2004). Adaptive Web Document Classification with MCRDR. International Conference on Information Technology: Coding and Computing ITCC 2004, Orleans, Las Vegas, Nevada, USA.

Liu, L., C. Pu, et al. (2000). WebCQ: Detecting and Delivering Information Changes on the Web. International Conference on Information and Knowledge Management (CIKM), Washington D.C., ACM Press.

Liu, L., W. Tang, et al. (2002). "Information Monitoring on the Web:A Scalable Solution." World Wide Web Journal **5**(4).

Mladenic, D. (1999). "Text-learning and Related Intelligent Agents." Applications of Intelligent Information Retrieval.

Park, S. S., S. K. Kim, et al. (2003). Web Information Management System: Personalization and Generalization. the IADIS International Conference WWW/Internet 2003.

Park, S. S., Y. S. Kim, et al. (2004). Web Document Classification: Managing Context Change. IADIS International Conference WWW/Internet 2004, Madrid, Spain.

Quan, T. T., S. C. Hui, et al. (2005). "A fuzzy FCA-based approach for citation-based document retrieval." 2004 IEEE Conference on Cybernetics and Intelligent Systems (IEEE Cat.04EX912): 578-83 vol.1.

Rajapakse, R. K. and M. Denham (2003). A Reinforcement Strategy for (Formal) Concept and Keyword Weight Learning for Adaptive Information Retrieval. MLIRUM'03: Second Workshop on Machine Learning, Information Retrieval and User Modeling at the Ninth International Conference on User Modeling, Pittsburgh, PA, USA.

Richards, D. (1998). The Reuse in Ripple Down Rules Knowledge Based Systems, University of New South Wales.

Richards, D. (2001). "Combining cases and rules to provide contextualised knowledge based systems." Modeling and Using Context. Third International and Interdisciplinary Conference, CONTEXT 2001. Proceedings (Lecture Notes in Artificial Intelligence Vol.2116): 465-469.

Sebastiani, F. (2002). "Machine learning in automated text categorization." ACM Computing Surveys **34**(1): 1-47.

Tang, W., L. Liu, et al. (2000). WebCQ Detecting and Delivering Information Changes on the Web. Proc. Int. Conf. on Information and Knowledge Management (CIKM).

Interaction Design for a Mobile Context-Aware System Using Discrete Event Modelling

Annika Hinze **Petra Malik** **Robi Malik**

Department of Computer Science, University of Waikato, Hamilton, New Zealand
E-mail: {hinze,petra,robi}@cs.waikato.ac.nz

Abstract

This paper describes our experience when applying formal methods in the design of the tourist information system TIP, which presents context-sensitive information to mobile users with small screen devices. The dynamics of this system are very complex and pose several challenges, firstly because of the sophisticated interaction of several applications on a small screen device and the user, and secondly because of the need for communication with highly asynchronous event-based information systems.

In a first step, UML sequence diagrams have been used to capture the requirements and possible interactions of the system. In a second step, a formal model has been created using discrete event systems, in order to thoroughly understand and analyse the dynamics of the system. By verifying general properties of the formal model, several conceptual difficulties have been revealed in very early stages of the design process, considerably speeding up the development. This work shows the limitations of typical methods for interaction design when applied to mobile systems using small screen devices, and proposes an alternative approach using discrete event systems.

1 Introduction

Interaction design for mobile location-aware systems is a very challenging task that can only to a limited extent draw from experiences in designing desktop-based systems (Kjeldskov, Graham, Pedell, Vetere, Howard, Balbo & Davies 2005). We are implementing TIP, a mobile tourist information system that provides context-aware information delivery to travellers. Currently, the software is being redesigned into a service-oriented architecture to support the system's various services in a flexible way.

The complex interactions between the TIP core system and its various services together with the management of user context (such as location and interests), and the interplay with the mobile device create several design challenges (Hinze & Buchanan 2005). Typically, the design of mobile systems focuses on interface issues, while the design of location-aware systems, such as tourist guides, often follows a user-centred design combined with a-posteriori usability studies.

In contrast, we follow a formal design approach to address the service interaction issues. We use automata for discrete event modelling (Cassandras & Lafortune 1999, Ramadge & Wonham 1989) since this design methodology mirrors the abstraction concept that guides the system's internal control using event-based interaction.

Traditionally, formal methods have mostly been applied to mission-critical software. This paper shows the advantages of using a formal approach in the design of a mobile location-aware system. We provide a complex formal model of the TIP system and show how its analysis leads to a much improved system architecture.

Section 2 provides a brief introduction to the TIP system and its architecture. Section 3 illustrates the design of the interactions within and between the services using UML sequence diagrams. Section 4 shows the interaction design by modelling interactions with automata for discrete event systems. Section 5 briefly refers to related work. In Sections 6 and 7, we conclude the paper with a discussion of the lessons learned regarding interaction design for mobile systems, and a summary of our findings and future work.

2 The TIP System

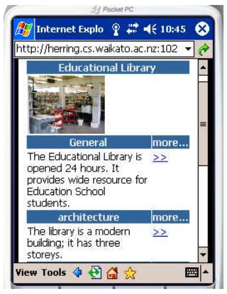
TIP is a mobile system that delivers tourist information about sights based on the user's context: their location, two personal profiles describing interests, and the user's travel history. The system also considers the sights' context, such as their location and memberships in semantic groups. Sights that are in a semantic group share certain features, e.g., medieval churches. Details about the TIP core can be found in (Hinze & Voisard 2003). In addition, TIP supports several services such as recommendations and a map service (Hinze & Junmanee 2005, Hinze & Buchanan 2006).

This section briefly illustrates TIP in use, argues for a new service-oriented architecture for TIP 3.0, and describes the basic interactions within the architecture.

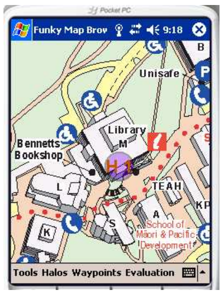
2.1 TIP in Use

Figure 1 shows the TIP core system and its services in use. In Figure 1(a), sight-related information is delivered for the current location of the user. TIP also supports navigation by maps that dynamically update the current location and the location of nearby sights. Figure 1(b) shows the map for the same location as before. For recommendations, we assume that a user who has seen several sights in a group is interested in seeing more (see Figure 1(c)). Recommendations are also given based on user feedback and profiles. Users may also browse for places they are not currently visiting—this is indicated by a different colour scheme as shown in Figure 1(d).

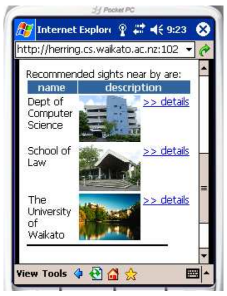
The TIP system combines an event-based infrastructure with a location-based service. The heart of the system is the filter engine cooperating with the location engine. The filter engine selects appropriate information from different source databases depending on user and sight context. Thus, the system's interaction logic is defined in context-aware profiles for the event-based filter engine. The system is implemented as a client/server architecture, supporting desktop clients as well as mobile clients on a hand-held device with appropriate interfaces.


Internet Explo 10:45
http://herring.cs.waikato.ac.nz:102
Educational Library
General more...
The Educational Library is opened 24 hours. It provides wide resource for Education School students. >>
architecture more...
The library is a modern building; it has three storeys. >>
View Tools

(a) Current location


Funky Map Brov 9:18
Unisafe
Library
Bennetts Bookshop
TEAH
School of Māori & Pacific Development
Tools Halos Waypoints Evaluation

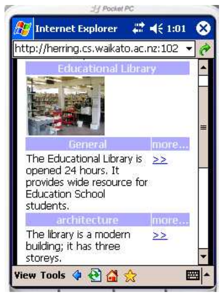
(b) Map


Internet Explore 9:23
http://herring.cs.waikato.ac.nz:102
Recommended sights near by are:

name	description
Dept of Computer Science	>> details
School of Law	>> details
The University of Waikato	>> details

View Tools

(c) Recommendations


Internet Explorer 1:01
http://herring.cs.waikato.ac.nz:102
Educational Library
General more...
The Educational Library is opened 24 hours. It provides wide resource for Education School students. >>
architecture more...
The library is a modern building; it has three storeys. >>
View Tools

(d) Browse location

Figure 1: TIP in Use: Context-dependent information delivery; core system information delivery (a), services (b and c), and browsing interface (d)

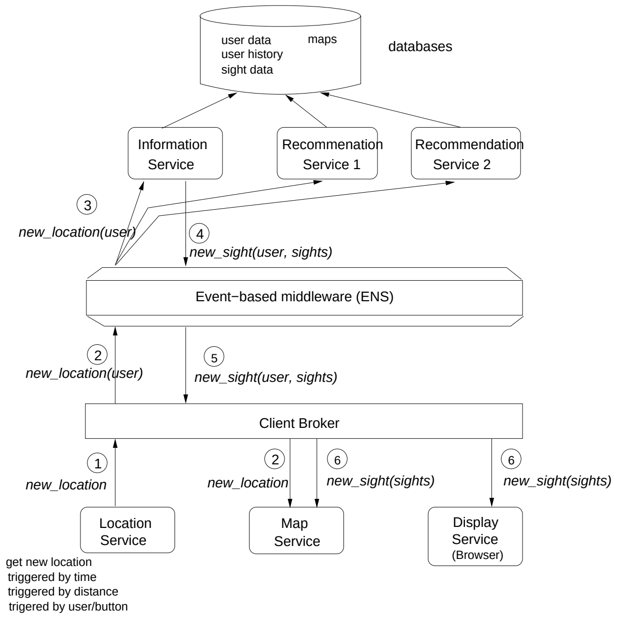


Figure 2: TIP 3.0 architecture and interactions between services and components

2.2 Towards a Service-Oriented Architecture

The current version 2.9 of TIP is a client/server system that is being extended to a hybrid client/server and peer-to-peer structure in TIP 3.0.

The apparently simple final service provided to the user by TIP 2.9 (a tourist guide with map, sight data, and recommendations) is in fact a composition of a variety of services (Hinze & Buchanan 2006). These services need to communicate with each other both within the same machine and between computers, using thick- and thin-client scenarios, and occasionally hybrid approaches. Existing mobile information systems often follow a monolithic approach. Our modular, service-oriented approach allows for the exploration of alternatives, e.g., of communication or implementation. Services can be provided in different ways or add new features without requiring changes to existing services.

With TIP 2.9, we have created a framework in which mobile services cooperate (e.g., the map and the information display system to provide sight indicators on the map). For TIP 3.0, further forms of cooperation and composition are needed. For example, we plan to extend the map to support different halo cues (e.g., for sight type or recommendation), which requires advanced inter-process communications.

With the trend of information systems increasingly being accessed using mobile devices and small screen interfaces, the aforementioned requirements will become more pronounced. Client devices will provide a number of pre-installed services and users will add their own selections. Consequently, we believe that for TIP 3.0 even stronger decoupling and modularisation is needed: a mobile infrastructure for mobile information services needs to flexibly support existing, changing, or new services. The design of TIP 3.0, for which the interaction design is reported in this paper, will see the redesigning TIP into a Service-Oriented Architecture (SOA) using web services.

2.3 Basic Interactions

Figure 2 illustrates the distributed nature of TIP's services and components. The databases and basic services are held on the server side (Figure 2, top). Each mobile client can have several services running. Interaction between services on the mobile device is coordinated by the client broker (Figure 2, bottom). The middle layer is formed by an event-based communication middleware. We abstract from details of the implementation on server side.

To clarify the architecture, we now explain the basic interaction sequence of the TIP system after a user's location change. This sequence is indicated using circled numbers ① to ⑥ in Figure 2; it is also specified by the sequence diagram in Figure 3.

First, the *location service* sends the current location of *this* user. This may be triggered by time (e.g., send every minute) or by location (e.g., send if the user moved more than 500m) or by user interaction with the system interface (e.g., a button being pressed to retrieve the new location). In Step ①, this information is sent to the client broker that coordinates all services on this mobile device.

In Step ②, the *client broker* attaches the user ID and sends the information to the event-based middleware (ENS). The information is also sent to the map service that is running locally on the mobile device. Here, we abstract from the possible distribution of the ENS over the mobile clients and the server. The *event-based middleware* holds service registrations and profiles about the information need of each service. As a basic service offer, it sends the information about the user's current location to the *information service* (and to all other services that requested this information). This is shown as Step ③. The event-profile logic that can be used for the ENS in this context is introduced in (Hinze & Voisard 2003). The information service receives the user's location (`new_location`) and the user ID; it looks up the sights nearest to the location that are of interest to the user.

As Step ④, the information service sends back to the ENS the list of sights that are close to the user (`new_sight`). The ENS forwards that information to the user's mobile device (i.e., to the client broker); shown as Step ⑤. The client broker distributes the information to the local services as needed, in Step ⑥. In this case, the *display service* receives the list of sights and sight information; it will display this information to the user to choose the sight in which they are interested.

The *recommendation service* also receives the user's location (`new_location`) and the user ID in Step ③. It looks up the sights that would be of interest to the user and

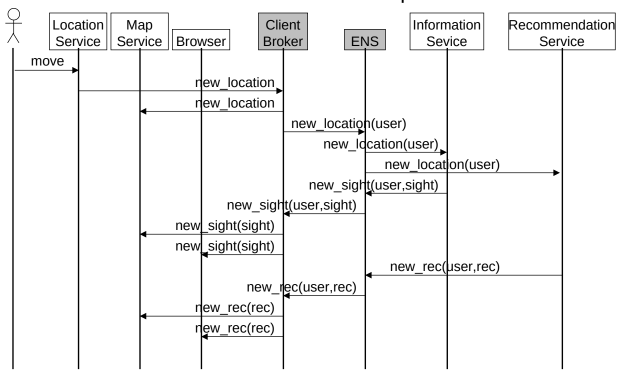


Figure 3: Basic interaction: User change of location. As the user moves, browser and map are updated with location, sights and recommended sights (shaded services are located on the middleware layer)

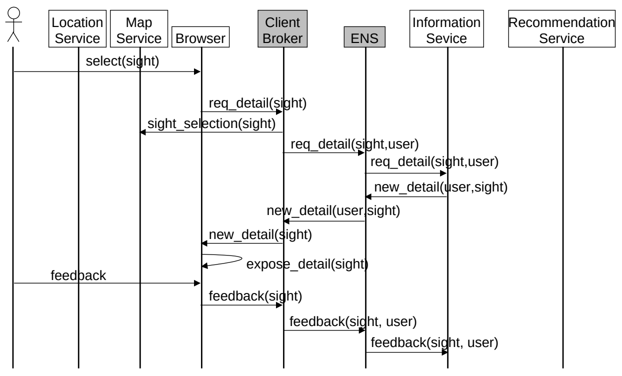


Figure 4: Select sight from the list for more information. Interaction follows after interaction in Figure 3

which the user has not seen already. Recommendations may also use feedback from other users. The functionality of the recommendation services is described in detail in (Hinze & Junmanee 2005). As shown in Figure 3, the recommendation service sends a list of sights back to the ENS. This list is then forwarded to the client broker on the mobile device. The client broker distributes the information to all services that are interested, i.e., the display service (for profiling information about the sights) and the map service (for highlighting the recommended sights).

In addition to this basic interaction, the TIP system needs to support several other interaction sequences. These are described in the next section.

3 Interaction Design Using Sequence Diagrams

This section describes several scenarios of possible interactions of the users and services in the TIP system. UML sequence diagrams (Kent 2001, Object Management Group 2003) are used, because they are a commonly understood means to describe interaction sequences. We focus on the interactions of a mobile client with the server, and abstract from the possible distribution of the data across other peers.

The scenarios presented in the following are grouped into basic interactions that describe the standard functionality of the TIP system, refined interactions that elaborate the details of how location information is to interpreted, and interface interactions that explore the specifics of user interaction on a mobile device.

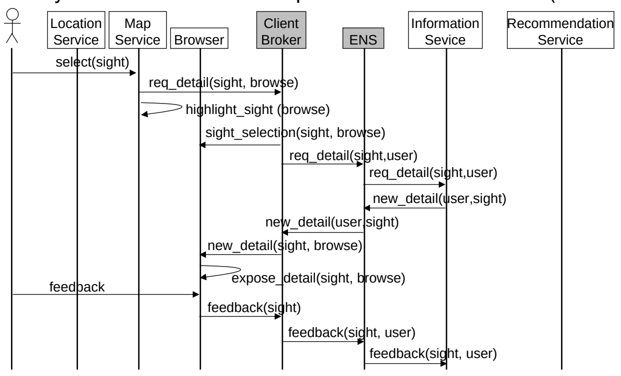


Figure 5: Select (distant) sight information, e.g., about recommended sights, from the map

3.1 Basic Interactions

1. **User change of location (see Figure 3):** The user changes their location and, subsequently, browser and map are updated with location, sights and recommended sights. This interaction is described in detail in the previous section.

2. **Select sight for more information (see Figure 4):** The user picks the sight they are looking at from the list of sights presented to them on the screen or from the map. If necessary, the application switches over to a browser interface when the information is available. Users can give feedback about how they liked the sight. Moving the mouse over the sight in the map without clicking could give some information (already downloaded in the task in Figure 3).

3. **Select sight information for distant sights / browse (see Figure 5):** An example is the browsing for more information about recommended sights (from the browser or the map). The design in Figure 5 (selecting on the map) has the drawback that people can assign feedback to sights they did not visit. Another option would be to present a list with all sights visited and not rated, and another with all sights visited to redo the ratings. The given design is useful to set up the system when used the first time—users can rate sights they know, even though they did not have the system yet.

4. **Browse for new locations on the map:** The user can also explore new areas of the map, placing a virtual user onto a new location, to receive information about that place without having to be at that location. Zooming and moving of the map is equivalent to scrolling in a browser; all interaction stays within the service. Location-browsing, as modelled here, is the equivalent of browsing on information pages as modelled in Task 3.

3.2 Refined Interactions

The modelling of location transmission can be refined to capture different approaches. This opens new possibilities for interaction design.

5. **New location transmission (see Figure 6):** So far, we abstracted from the different modes of sending locations. As indicated in Figure 2, location is measured and transmitted constantly (we ignore here the internal control of the location signal, such as GPS); the transmission of the location information to other services depends on the mode: location can be sent (a) continuously whenever the user moves, (b) depending on time since the last reading, (c) depending on distance from last reading, and (d) triggered

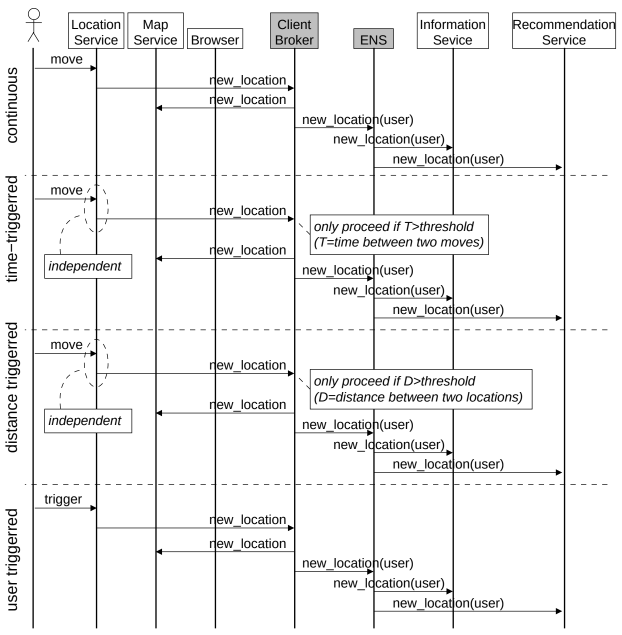


Figure 6: Different variations for location transmission

by user action (e.g., button pressed). Hybrid forms are also conceivable and depend on the actual location service implementation (e.g., location signal is sent according to time frequency or if the distance crosses a threshold). In fact, the system may have to behave differently for different services involved (see Task 6). Currently, TIP 2.9 supports the latter mode of requiring user interaction with the system, which leads to several irritating features for the user interaction.

Modelling the different modes using sequence diagrams does not reveal their severe consequences for user interactions (such as a divergence between screen location and real location of the user without indication in the interface).

When the user moves, their representation (e.g., avatar or location-halo) in the map should move as well. This does not hold for the information delivered: We assume that new information is only shown when triggered by the user to avoid flickering of the display. Also, when information about a building is received, the user should be able to access this information while walking around the building.

6. **Browsing by walking (see Figure 7)**: This action depends on the implementation of Task 5 (transmission of location data). Switching into a different mode, users are able to browse through their environment by walking. On their screen, new locations and sights are continuously updated. Consequently, the interaction mode has to change for both interfaces (map and browser) when switching into this particular behaviour. One possible sequence diagram is shown in Figure 7: location information is continuously sent; the interfaces are updated whenever new information is available; the information system display is only updated on request. This mode is not designed for reading and browsing through content but for scanning the environment and discovery of new places.

3.3 Interface Interactions

The interactions for navigating on the screen and in the information history cannot be modelled satisfactorily using

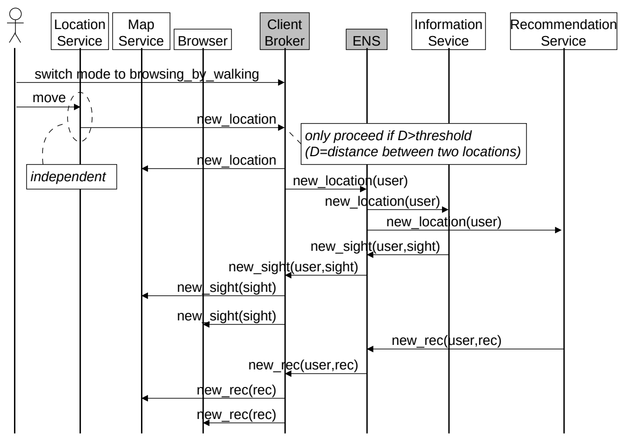


Figure 7: Browsing by walking

sequence diagrams. The different states of the interface are an important aspect that is missing in these models.

7. **Interaction: To current location (Synchronise)**: On pressing this button, the user interface returns to the current location of the user on the map. The location is newly measured and sent to the system; that is the user and the system are synchronised. This button is helpful if the user was browsing through information or zooming and navigating on the map.

8. **Interaction: To last location ('Back')**: On pressing this button, the user interface returns to the last location that was measured, ignoring user moves in between. This gives the user the opportunity to return to the last information that was delivered before they started browsing and/or walking around. It also offers the option of moving backwards through locations instead of presented pages.

9. **Interaction: Navigation 'Back'/'Forward'**: On pressing this button, the user can browse backwards through the pages that have previously been presented to them, or move forward from earlier pages. This navigation should work for all services (each service itself may provide their own navigation).

The above requirements show that the user interface has to provide options for setting different modes, and several navigation options to support different kinds of histories in this location-aware system.

4 Interaction Design Using Discrete Event Modelling

While the message sequence diagrams developed in the previous section provide a good description of the requirements and possible usage scenarios of the TIP system, they cannot show the dynamics of the system behaviour and the complex interactions with the user interface. In contrast to modelling for an immobile desktop system, the small-screen interface on a mobile device considerably influences the system behaviour. Together with the necessary interactions with all the other services and the location-related aspects, this leads to very sophisticated concurrent interactions that need to be modelled carefully.

4.1 The Missing Link: Interfaces and States

Douglas (Douglas 1999) distinguishes between simple, state, or continuous behaviour of objects in systems modelling. We identify the dynamic behaviour of the services in the TIP system as state-full. Therefore, finite-state machines are an appropriate technique to model these concurrent interactions. UML supports the modelling of finite-state machines with UML statecharts and UML activity diagrams (Object Management Group 2003, Douglas 1999). While UML statecharts have been used successfully to design reactive and real-time systems, there still is only limited tool support for the formal analysis of complex statecharts models.

We strongly believe that, when modelling concurrent activities, the designer needs to be able to interact with the system to explore its capabilities and pitfalls. We therefore do not use UML statecharts, but a modelling tool for discrete event systems, which allows us to analyse the model for conflicts, and allows for interaction and playful exploration of the modelled system. The relationship between UML statecharts and discrete event systems, and a possible translation between the two formalisms is explored in (Malik & Mühlfeld 2003).

4.2 Principles of Discrete Event Modelling

We use the VALID Toolset to model and analyse the behaviour of the TIP client. The VALID Toolset, developed at Siemens Corporate Technology, supports the modelling, verification, and code generation of finite-state automata as described in (Brandin, Malik & Malik 2004, Malik & Mühlfeld 2003). It uses the modelling methodology of discrete event systems (Cassandras & Lafortune 1999, Ramadge & Wonham 1989).

In this framework, concurrent systems are modelled using several *finite-state automata* running in parallel. Each automaton is represented graphically as a state transition graph as shown in Figure 8. States are represented as nodes, with the initial state highlighted by a thick border, and terminal states coloured grey. Transitions are represented as labelled edges connecting the states: in Figure 8(c), e.g., the automaton `map_expose` changes its state from $hidden$ to $exposed$ with an occurrence of the event map_expose. If an edge is marked with more than one label, this indicates that any one of the corresponding events causes the transition. Automaton `map_expose` in Figure 8(c), e.g., changes from state $exposed$ to state $hidden$ if a br_expose or a map_hide event occurs.

If a certain state does not have any outgoing transitions labelled with a certain event, that event is *disabled* by the automaton and cannot occur in the system. For example, automaton `map_expose` in Figure 8(c) disables event map_expose when in state $exposed$. This rule applies only to events that occur somewhere in an automaton or, more precisely, to events that constitute the so-called *event alphabet* of the automaton. Events that do not occur in the event alphabet remain enabled and do not cause any state change of the automaton when they occur. For example, event map_show(1) occurs in the event alphabet of `map_new[1]` in Figure 8(b), but not in `map_expose` in Figure 8(c). Therefore, automaton `map_expose` is always ready to perform a transition with event map_show(1), leaving its state unchanged.

In this way, several automata are composed by *synchronisation on common events*. All synchronised automata repeatedly agree on an event to be executed next, and simultaneously perform the corresponding state transition. A state transition using an event σ can only take place if all synchronised automata that have σ in their event alphabet allow the event σ to occur.

For more details on synchronous composition and other concepts from the theory of discrete event systems, please refer to (Cassandras & Lafortune 1999, Ramadge & Wonham 1989). The benefit of this very simple framework is that it does not only enable us to create and simulate a concise model of the dynamic system behaviour, but also makes it possible to use *model checking* (Clarke, Grumberg & Peled 1999). This ability to check critical properties of the model quickly and automatically has helped us to discover several problems.

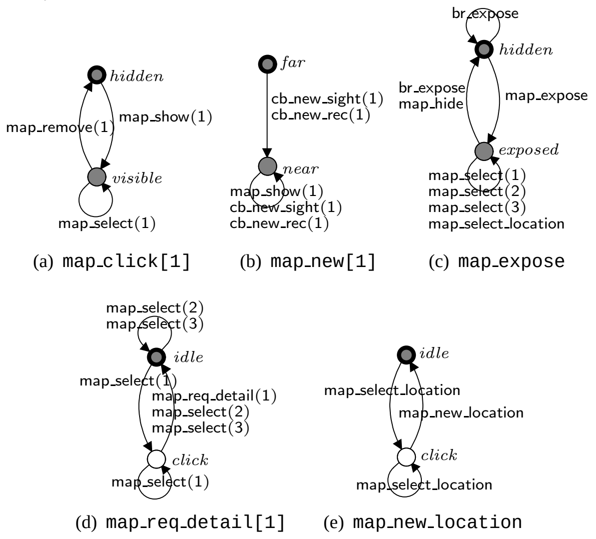


Figure 8: The map model

4.3 A Model of the TIP Client

Our model of the TIP system includes all the components constituting the TIP client on a mobile device as shown in Figure 2, namely the location service, the map service, the browser, and the client broker. It also includes a model of the behaviour of the event-based middleware (ENS), to the extent needed to capture the communication between the client and the ENS. The details of the information systems on the server side are not modelled.

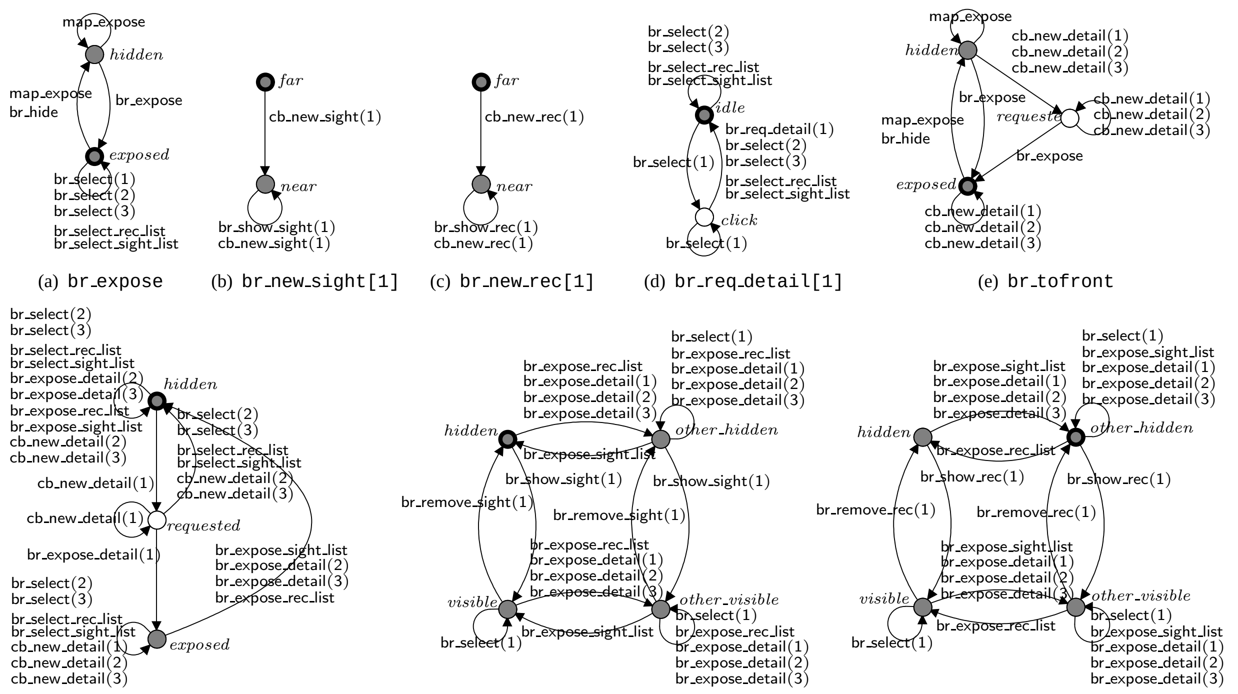
When modelling a system that includes large amounts of complex data, such as TIP, using discrete event systems, abstractions are necessary in order to guarantee that the state space remains finite. In a discrete event model, it usually is not possible to include details about the information attached to the various events that may occur—only the fact that a message is sent or received is modelled, in most cases completely abstracting away from the associated data.

In some cases, a certain reference to the data is needed to retain an interesting aspect of possible interactions. In our model, we assume that there are three different Sights 1, 2, and 3 that can be displayed. This number guarantees a tractable model, and is completely sufficient to reveal conceptual problems in the design. For clarity of the model, we also used other small simplifications.

In the following, we describe each of the service components separately. The next section explains the interplay of these components, and shows some of the results obtained from the formal analysis of the model. A more detailed model can be found in (Hinze, Malik & Malik 2005).

The Map Service. The map service shows the location of the user as well as sights and recommendations nearby. For the purpose of modelling the map service, sights and recommendations are treated alike, although they may be displayed differently on the screen. Figure 8 shows a simplified model of the map service, which only supports a single map with a fixed number of up to three sights or recommendations. Dynamic updates of the map area are not supported, nor is the removal of sight information, nor the highlighting of the sight shown in the browser.

Since the model is restricted to three sights, the map can show none, one, two, or three sights. Automaton `map_click[1]` in Figure 8(a) models whether Sight 1 is shown on the map or not. There exist similar automata for Sights 2 and 3, which are not depicted in Figure 8 since the only difference is the sight number in brackets.



(a) `br_expose` (b) `br_new_sight[1]` (c) `br_new_rec[1]` (d) `br_req_detail[1]` (e) `br_tofront`

(f) `br_expose_sight[1]` (g) `br_click_sight[1]` (h) `br_click_rec[1]`

Figure 9: The browser model

Initially, Sight 1 is not shown. Becoming visible is modelled by event map_show(1), and becoming invisible is modelled by event map_remove(1), both changing the state of automaton `map_click[1]`. The automaton uses another event, map_select(1), which does not change the state of the automaton. The important point here is that event map_select(1) is only possible when Sight 1 is shown on the map; an obvious consequence of how users can interact with their device.

The map highlights known sights and makes them selectable for the user. Automaton `map_new[1]` in Figure 8(b) models that Sight 1 is shown on the map after the map has received information about it as a sight (event cb_new_sight(1)) or as a recommendation (event cb_new_rec(1)) from the client broker. Again, there are similar automata for Sights 2 and 3 not depicted here. Also note that automata `map_new[1]` and `map_click[1]` synchronise on the common event map_show(1), i.e, the event map_show(1) is only possible if both automata are in a state where this event is allowed.

There are more factors that influence whether it is possible to click at the map browser. The map can be either exposed, which means it is visible on the screen, or hidden, and thus not visible on the screen. The user is only able to click onto the map if it is exposed. This behaviour is modelled by automaton `map_expose` in Figure 8(c).

Automata `map_req_detail[1]` in Figure 8(d) and `map_new_location` in Figure 8(e) describe the behaviour of the map if the user clicks on it. The user can select sights displayed on the map to request for detailed information, or click elsewhere to explore other areas. After one or possibly more clicks on Sight 1, the map sends a map_req_detail(1) event to the client broker, requesting detailed information to be shown in the browser. The attempt to send this message is cancelled if the user clicks on another sight. If the user clicks on the map to browse to a different location (event map_select_location), a map_new_location event is sent to the client broker in order to start searching for sights and recommendations close to the selected location.

The Browser. The browser is used to show textual information about sights visited by the user. In the model, it can display a list of sights close to the current position of the user, a list of recommendations close to the current position of the user, or the detailed information of a specific sight. It can display information in the form of web pages received from the client broker, and request detail information, if the user clicks on hyperlinks. The current model assumes an intelligent browser that can combine information from several messages and change the display accordingly.

When a sight or recommendation gets close enough, the browser receives a cb_new_rec(1) or cb_new_sight(1) message from the client broker. In response to this, the browser updates its sight or recommendation list, indicated by event br_show_rec(1) or br_show_sight(1), as modelled by automata `br_new_rec[1]` in Figure 9(c) and `br_new_sight[1]` in Figure 9(b).

If the user clicks on a hyperlink to request details about Recommendation or Sight 1 (event br_select(1)), a br_req_detail(1) event is sent to the client broker unless the user clicks onto another recommendation or sight before that event could be sent. This is modelled in automaton `br_req_detail` in Figure 9(d) and works like the processing of clicks on the map. The client broker responds to the br_req_detail(1) event by sending a cb_new_detail(1) event, providing a web page with detailed information about the sight. As soon as this message arrives, the browser is exposed (unless it is already, see automaton `br_tofront` in Figure 9(e)), and the information about the sight is displayed (see automaton `br_expose_sight` in Figure 9(f)).

A more detailed description of all automata in Figure 9 is given in (Hinze et al. 2005).

The Client Broker. The client broker links the applications on the mobile device with the network and the event notification system (ENS) on the server side. It also coordinates the map display and browser to ensure that control is passed correctly between these two applications.

In most cases, the client broker simply forwards events received from the applications to the ENS and vice versa.

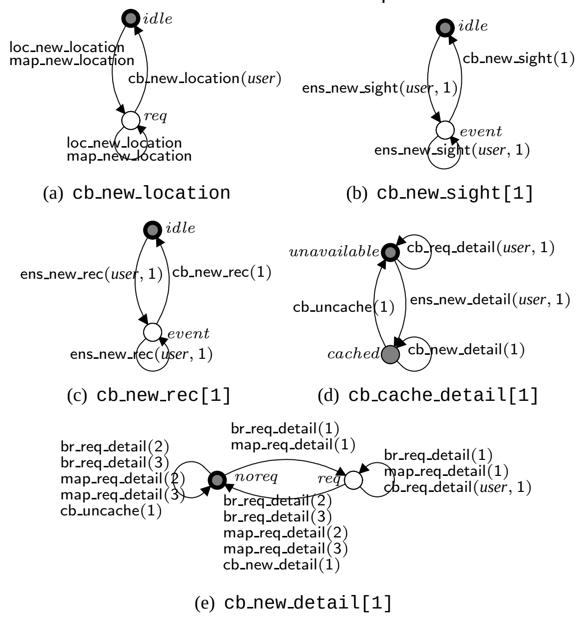


Figure 10: The client broker model

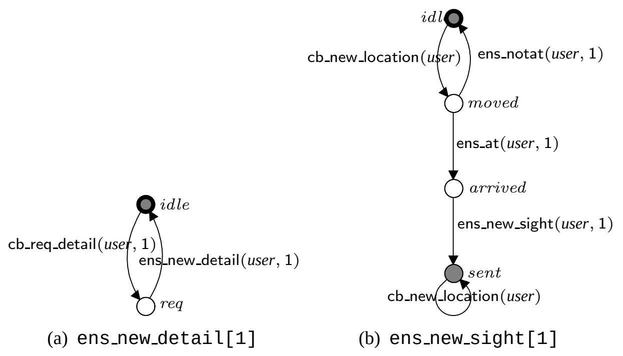


Figure 11: The ENS model

For example, automaton `cb_new_location` in Figure 10(a) forwards location changes from the client's location server (event loc_new_location) or from the map (event map_new_location) to the ENS after attaching the user ID. Likewise, automata `cb_new_sight` in Figure 10(b) and `cb_new_rec` in Figure 10(c) model how information from the ENS about new sights or recommendations, respectively, is sent back to the map and browser.

The model for detailed sight information, as given by automata `cb_new_detail[1]` in Figure 10(e) and `cb_cache_detail[1]` in Figure 10(d), is more elaborate because the client broker also has the ability to cache some of the web pages received. The browser or map can request detailed information about Sight 1 by sending a br_req_detail(1) or map_req_detail(1) event, which causes the automaton `cb_new_detail[1]` to enter state req. In this state, the client broker is ready to forward the request augmented with the user ID to the ENS (event cb_req_detail$(user, 1)$). This event is answered by the ENS by providing detailed information (event ens_new_detail$(user, 1)$), which is stored in the cache, indicated by automaton `cb_cache_detail[1]` changing into state $cached$. At this stage, the client broker can send a cb_new_detail(1) event to the browser, which will display the detailed information.

The Server Side. The model of the server side states the assumption that all requests sent to the ENS are actually answered. Automaton `ens_new_detail` in Figure 11(a) shows that a request for detailed information in form of a cb_req_detail$(user, 1)$ event is answered by an ens_new_detail$(user, 1)$ event with the corresponding web page. Furthermore, the information servers respond to location changes of the user, signalled by cb_new_location$(user)$ events. If such an event occurs, the databases are searched for sights that are close to the new position. The result of the search is modelled by the two events ens_notat$(user, 1)$ and ens_at$(user, 1)$. If Sight 1 is close enough (event ens_at$(user, 1)$), this information is sent as an ens_new_sight$(user, 1)$ event to the client broker. Similar automata are used for recommendations.

Table 1: Example trace corresponding to Figure 3

	`cb_new_location`	`ens_new_sight[1]`	`cb_new_sight[1]`	`map_new[1]`	`br_new_sight[1]`
(initial state)	$idle$	$idle$	$idle$	far	far
loc_new_location	req	$idle$	$idle$	far	far
cb_new_location$(user)$	$idle$	$moved$	$idle$	far	far
ens_at$(user, 1)$	$idle$	$arrived$	$idle$	far	far
ens_new_sight$(user, 1)$	$idle$	$sent$	$event$	far	far
cb_new_sight(1)	$idle$	$sent$	$idle$	$near$	$near$
map_show(1)	$idle$	$sent$	$idle$	$near$	$near$
br_show_sight(1)	$idle$	$sent$	$idle$	$near$	$near$

4.4 Analysing the Model

Having created the model, it is possible to run simulations and check whether the model can perform the interactions specified by the sequence diagrams. For example, Table 1 shows the event sequence corresponding to the basic interactions following a location change as specified in Figure 3.

When a location change occurs, the location server on the mobile device sends a loc_new_location event to the client broker. This event triggers the automaton `cb_new_location` to send a cb_new_location$(user)$ event to the ENS. The ENS, in this case represented by automaton `ens_new_sight`, may now detect that the user is close to Sight 1 (event ens_at$(user, 1)$) and respond by sending an ens_new_sight$(user, 1)$ event. This message in turn triggers automaton `cb_new_sight`, which forwards the event as cb_new_sight(1) to the map service and browser. The cb_new_sight(1) event simultaneously activates automaton `map_new` in the map service and `br_new_sight` in the browser, which causes them to display the new sights by events map_show(1) and br_show_sight(1), respectively.

In the discrete event model, a single cb_new_sight(1) from the client triggers actions both by the map service and the browser. The same communication is modelled by two separate interactions in the message sequence diagram in Figure 3. The notion of the client broker sending a single event, without necessarily knowing the receivers, seems to capture the intended behaviour of the event-based middleware more accurately.

In addition to simulating the possible behaviours of the model, it is possible to perform some formal analysis and check whether the model satisfies properties of interest. We have checked that the model is *controllable* (Cassandras & Lafortune 1999, Ramadge & Wonham 1989), i.e., that it is always ready to react to any possible events caused by user interactions or the information systems, and that it is *nonblocking* (Cassandras &

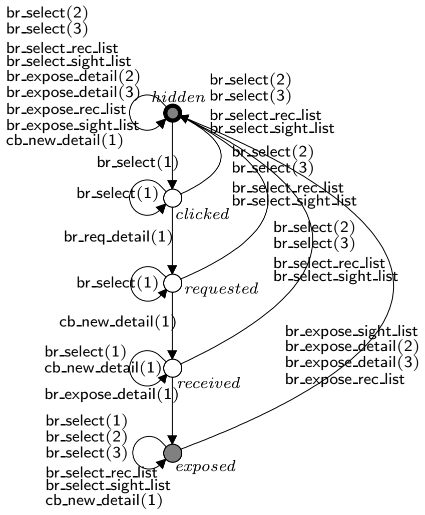


Figure 12: Broken version of `br_expose_sight[1]`

Lafortune 1999, Ramadge & Wonham 1989), i.e., that it is free from deadlocks and livelocks.

These checks did not succeed immediately. By repeatedly verifying properties, we discovered several problems in the model. Then we eliminated the problems and checked the model again until all critical properties could be checked.

An earlier version of the model included the automaton in Figure 12. It represents an attempt at creating a smarter browser that tries to avoid showing sight details if they are not the last thing the user clicked on. The idea is that, if the user clicks a link to Sight 1 in the browser, the browser requests detailed information about this sight from the client broker. However, this request is cancelled if the user clicks on something else before processing can be completed.

Formal analysis shows that this approach does not work. If we replace automata `br_req_detail[1]` in Figure 9(d) and `br_expose_sight[1]` in Figure 9(f) by the alternative in Figure 12, and make similar changes for Sights 2 and 3, the model contains a deadlock and therefore cannot be proven to be nonconflicting. When trying to verify this property, the VALID Toolset produces a counterexample that reveals a problem in the design.

The problem is as follows. The user may click on a link to Sight 1 (event br_select(1)), which causes the browser to send a request for details about Sight 1 to the client broker (event br_req_detail(1)). At this stage, automaton `br_expose_sight[1]` in Figure 12 is in state *requested* and waits for the client broker to send detail information about Sight 1. But before this information arrives, the user may activate the map (event map_expose) and click on sight two on the map (event map_select(2)). As a result, the map server sends a request for details about sight two to the client broker (event map_req_detail(2), see Figure 8(d)). This causes the client broker to cancel the request for details about Sight 1, and only to process the second request (see Figure 10(e)). As a result, the browser will never receive information about Sight 1, i.e., the automaton in Figure 12 will never receive the cb_new_detail(1) event needed to leave state *requested*.

The root of this problem is that the browser cannot accurately determine which is the most recent request from the user. The browser only knows about user interactions with the browser, not with the map service. In consequence, the decision about the most recent request must be shared between the two applications. To overcome the problem, the model has been changed to the one presented in Figure 9, where the client broker determines which request is the most recent by checking the events received from the browser and map service. The browser merely displays all the detail pages received.

After fixing all problems, the model has been verified successfully by the VALID model checker. All checks can be completed in three minutes on a 1.4 GHz Athlon processor with 256 MB of RAM. This shows that the finite-state automata model already is fairly complex and serves as a challenging benchmark for model checking tools.

5 Discussion of Related Work

This work touches on the research of the systems design and HCI communities. We also consider typical approaches for modelling and evaluating location-aware mobile systems.

System Modelling and Design. Modelling of concurrent systems using state machines has chiefly been used for mission critical applications, often focusing on safety issues, e.g., flight interface control (Degani, Heymann, Meyer & Shafto 2000). Typical problems that have also been identified in our study are mode awareness, mode confusion, and automation surprise. Bolignano et al. (Bolignano, Le Métayer & Loiseaux 2000) call the application of formal methods in practical applications a 'missing link'.

For the design of interaction and interfaces for virtual environments, CSP (Communicating Sequential Processes) has been used (Smith, Marsh, Duke, Harrison & Wright 1998, van Schooten, Donk & Zwiers 1999). CSP is an alternative modelling principle which we could have used for our system. CSP is related to our modelling approach, but typically lacks modelling environments that support active simulation. In CSP applications, the aspects of location-awareness and mobile users are rarely considered. In virtual environments (as the ones mentioned above), the mobility of the user's avatar does not interfere with the location of the real user as is the case in our application.

User and Interaction Modelling. HCI research has focused on modelling of users and their interaction with a system. Blandford et al. evaluate different programmable user modelling (PUM) strategies for user-centred criteria such as tractability and re-usability (Blandford, Butterworth & Curzon 2004). Within their schema of operational and abstract models, our approach follows an abstract functional model, with the main advantage of provability. (Thimbleby, Cairns & Jones 2001) proposes modelling push-button devices using Markov models; this approach is similar to ours but evaluates a different kind of application. Degani et al. (Degani, Shafto & Kirlik 1999) address the problem of mode confusion using statecharts and modelling structures; their findings are more concerned with user interface design. Their modelling focuses on concurrency issues. The problem of mode confusion has also been addressed using formal modelling methods (Rushby 2002, Butler, Miller, Potts & Carreño 1998). These approaches chiefly deal with the awareness of the users of different modes (focusing on opacity, complexity, incorrect mental model) and on safety (one of the typical application fields of formal methods). The focus of the cited works is different to ours; in their evaluations, interactions between system components are not considered, nor are concurrency and synchronisation problems. In addition, the issues of location and user movements introduce challenges that are not addressed.

Design and Evaluation of Location-Aware Mobile Systems. Typical design and evaluation uses methods from desk-bound computers (Kjeldskov et al. 2005). The focus lies on the interface design of small screen devices and user-device interactions. Interaction design that takes location-awareness and mobility into account is rare.

In the distinction between user-centred and technology-centred design (Kjeldskov & Howard 2004, Brown 1998), our approach is closer to technology-centred design. We use reflection about an existing prototype and technology-driven service models, enriched with usage scenarios and enactment of future scenarios. The typical user-centred approach is followed in (Pousman, Iachello, Fithian, Moghazy & Stasko 2004): questionnaires, interviews and discussion with potential users lead to system design with subsequent user-testing of a functional prototype. The findings of this project underline that design for mobile hand-held devices needs to be distinct from design for desktop machines. One result is the necessity to easily restart or resume actions that have been interrupted. We address this issue by the introduction of different navigation methods in Section 3.3, which give users easy access to their current and previous states within the system interaction, and by actively propagating context changes to all services. Our design addresses selected issues (interaction syntax, resuming tasks, integration) identified in (Pousman et al. 2004) in an a-priori manner. Early HCI considerations for location-aware applications have been discussed in (Brown 1998); in our redesign process, we identify and address similar issues.

A-posteriori Usability Studies. Usability and interaction issues of mobile location-based systems have often been evaluated a-posteriori, that is after the implementation of the software (Iacucci, Kela & Pehkonen 2004). Kjeldskov et al. (Kjeldskov et al. 2005) point out that the special characteristics of tourist guides make the design and evaluation a challenging task: close relation to physical location and objects in immediate user surroundings as well as the ambulating user constantly changing physical location. They identified as critical usability issues those related to the mapping issues from the use of the system in the real world, such as (1) representation of information and interaction between user and system depending on the surrounding environment and (2) disparities in the relationship presented in the system and the context in which the user was situated. These are two of the issues that we also identified as critical in our design phase. Thus, critical issues typically observed in usability studies could be addressed already in the design phase.

Summary. Traditionally, formal modelling has mainly been used to address issues of concurrency, safety, and correctness. Mobility and location-awareness usually is not addressed directly. When addressing interaction design for mobile systems, typically only the aspects of user interaction are considered, since this community mainly regards problems of interactions using small screen displays. Location-aware mobile applications pose more challenges such as identity of locations, virtual/real-world confusion, complex interactions with cooperating services on the same device. When designing mobile location-aware applications, scenario and prototype-based approaches prevail. Only a few publications have dealt with the modelling of location-aware systems. Most evaluations are a-posteriori usability studies.

6 Lessons Learned

From our experience with modelling the system, we have identified the following issues. A clear distinction is needed between three user identities:

1. *The real-world user.* This is the person using the system; they can change their location in the real world. This location is captured by the location service in the system and triggers various responses.
2. *The virtual user.* This is the representation of the real-world user within the system. Typically the virtual user follows the movements of the real world user. The virtual user can also be used to 'visit' places within the system on behalf of the user (e.g., by clicking on the map to explore places at a distance). It should be possible to move through the location history of the virtual and the real user (e.g., using user-defined location points).
3. *The interacting user.* Independent of the location of the user, the interface allows to select browser pages, pictures and maps. The interaction is controlled by the real-world user. It should be possible to move through the interaction history with the system (similar to the 'Back' and 'Forward' buttons in a browser).

Several issues for the design of interactions between services, and between user and system have been identified. These issues occur as an immediate consequence of the mobility and location awareness on the system. They are typical for this kind of system.

The first one is mode confusion regarding the different users and places (e.g., location of the real user on the map while browsing with the virtual user). The user interface has to address this issue in a clear and comprehensible manner. This is especially important for the navigation through interaction and location histories. On the other hand, the server and involved services have to address this issue as well. For example, for deciding whether to include a certain location into the user history, whether to allow the user to give feedback about a place, or which information to display.

The second issue is location capturing. Several types of location capturing are supported for mobile systems; all provide varying semantics of new-location data (e.g., location measured after a time interval or depending on movement). These different types influence how the system can react, and how location events should be handled. A related issue is that of user movement in interaction with location capturing. This may lead to confusion about the actual location of the real-world user.

A third observation is that the context of the user has to be carried as explicit information, that is, the system has to react differently depending on its current context. This has been expressed earlier as 'modes', but that concept does not go far enough. Context is more complex than simple modes. Concepts from event-based systems with states may be explored here.

The fourth issue was identified when reasoning about the resulting model: System components may themselves have to be distributed (i.e., parts of the component reside on the client and other parts on the server). The current model assumes a certain part of the programming logic to reside on the client device (e.g., to change the display modes depending on the context). This may not be feasible for third-party thin clients such as standard web browsers displaying information from server-side JSP pages. This creates a major problem for re-usability of existing services and interfaces.

7 Conclusions

This paper presents the lessons learned during the redesign of a mobile location-aware system, using a combination of UML message sequence diagrams and discrete event systems. It proposes an improved service-oriented architecture and points out that different contexts need to be distinguished with great care in mobile location-aware systems, with implications not only for the user interface but also for the interaction protocols between the various services involved.

The formal modelling approach has greatly improved the system design. By describing the requirements formally, developers are forced to think carefully about the planned system. The activity of specifying and modelling on its own helps to understand and clarify many aspects of the system to be constructed.

We have seen that UML message sequence diagrams are very useful to describe particular sequences of actions, but fail to reveal problems that occur when several such action sequences are running together at the same time. Therefore, a finite-state automata model of the system has been created and analysed. The result is an abstract model of the *behaviour* of the system, which makes it possible to simulate the interaction between the various components and gain a thorough understanding of the dynamics of the system.

Exhaustive simulation and analysis of the model allows for finding subtle problems at this very early stage of the design. Several issues with the proposed interaction protocols have been discovered and solved, which may otherwise have remained undetected until much later during the implementation of the system.

As a next step for the interaction design, we plan to address the issues of interaction with other mobile peers and distribution of system components. For the overall system design, we plan to (1) support more services to identify common abstractions for a service oriented architecture; (2) include trust-based information exchange; and (3) model different applications with different services but the same base architecture. For the design process using discrete event systems, we plan to implement an interface simulation to allow for more direct user interaction with the simulated system.

References

Blandford, A., Butterworth, R. & Curzon, P. (2004), ‘Models of interactive systems: a case study on programmable user modelling’, *Int. J. Hum.-Comput. Stud.* **60**(2), 149–200.

Bolignano, D., Le Métayer, D. & Loiseaux, C. (2000), Formal methods in practice: the missing link. a perspective from the security area, *in* ‘Modeling and Verification of Parallel Processes (MOVEP 2000)’.

Brandin, B. A., Malik, R. & Malik, P. (2004), ‘Incremental verification and synthesis of discrete-event systems guided by counter-examples’, *IEEE Trans. Contr. Syst. Technol.* **12**(3), 387–401.

Brown, P. (1998), Some lessons for location-aware applications, *in* ‘Proc. 1st Workshop on HCI for Mobile Devices’, pp. 58–63.

Butler, R., Miller, S., Potts, J. & Carreño, V. A. (1998), A formal methods approach to the analysis of mode confusion, *in* ‘AIAA/IEEE Digital Avionics Systems Conf.’.

Cassandras, C. G. & Lafortune, S. (1999), *Introduction to Discrete Event Systems*, Kluwer.

Clarke, Jr., E. M., Grumberg, O. & Peled, D. A. (1999), *Model Checking*, MIT Press.

Degani, A., Heymann, M., Meyer, G. & Shafto, M. (2000), Some formal aspects of human-automation interaction, Technical Report NASA/TM-2000-209600, NASA Ames Research Center, Moffett Field, CA.

Degani, A., Shafto, M. & Kirlik, A. (1999), ‘Modes in human-machine systems: Constructs, representation, and classification’, *Int. J. Aviation Psychology* **9**(1), 125–138.

Douglas, B. P. (1999), ‘UML statecharts’, *Embedded Systems Programming* **12**(1).

Hinze, A. & Buchanan, G. (2005), Context-awareness in Mobile Tourist Information Systems: Challenges for User Interaction, *in* ‘Proc. Workshop on Context in Mobile HCI, in conjunction with Mobile HCI’, Salzburg, Austria.

Hinze, A. & Buchanan, G. (2006), The challenge of creating cooperating mobile services: Experiences and lessons learned, *in* ‘Proc. 29th Australasian Computer Science Conf. (ACSC 2006)’, Hobart, Australia. To appear.

Hinze, A. & Junmanee, S. (2005), Providing recommendations in a mobile tourist information system, *in* ‘Information Systems Technology and its Applications, 4th Int. Conf. (ISTA 2005)’, Palmerston North, New Zealand.

Hinze, A., Malik, P. & Malik, R. (2005), Towards a TIP 3.0 service-oriented architecture: Interaction design, Technical Report 08/05, Dept. of Computer Science, University of Waikato.

Hinze, A. & Voisard, A. (2003), Location- and time-based information delivery in tourism, *in* ‘Advances in Spatial and Temporal Databases (SSTD 2003)’, Vol. 2750 of *LNCS*, Satorini Island, Greece.

Iacucci, G., Kela, J. & Pehkonen, P. (2004), ‘Computational support to record and re-experience visits’, *Personal Ubiquitous Comput.* **8**(2), 100–109.

Kent, S. (2001), *Formal methods for distributed processing: a survey of object-oriented approaches*, Cambridge University Press, New York, NY, USA, chapter The unified modeling language, pp. 126–152.

Kjeldskov, J., Graham, C., Pedell, S., Vetere, F., Howard, S., Balbo, S. & Davies, J. (2005), ‘Evaluating the usability of a mobile guide: The influence of location, participants and resources’, *Behaviour and Information Technology* **24**(1), 51–65.

Kjeldskov, J. & Howard, S. (2004), Envisioning mobile information services: Combining user- and technology-centered design., *in* ‘Proc. Asia-Pacific Conf. Human-Computer Interaction (APCHI 2004)’.

Malik, R. & Mühlfeld, R. (2003), ‘A case study in verification of UML statecharts: the PROFIsafe protocol’, *J. Universal Computer Science* **9**(2), 138–151.

Object Management Group (2003), ‘Unified modelling language (UML), version 1.5’. Available at `http://www.omg.org`.

Pousman, Z., Iachello, G., Fithian, R., Moghazy, J. & Stasko, J. (2004), ‘Design iterations for a location-aware event planner’, *Personal Ubiquitous Comput.* **8**(2), 117–125.

Ramadge, P. J. G. & Wonham, W. M. (1989), ‘The control of discrete event systems’, *Proc. IEEE* **77**(1), 81–98.

Rushby, J. (2002), ‘Using model checking to help discover mode confusions and other automation surprises’, *Reliability Engineering and System Safety* **75**(2), 167–177.

Smith, S., Marsh, T., Duke, D., Harrison, M. & Wright, P. (1998), Modelling interaction in virtual environments, *in* ‘Proc. UK-VRSIG ’98’, Exeter, UK.

Thimbleby, H., Cairns, P. & Jones, M. (2001), ‘Usability analysis with markov models’, *ACM Trans. Comput.-Hum. Interact.* **8**(2), 99–132.

van Schooten, B., Donk, O. & Zwiers, J. (1999), Modelling interaction in virtual environments using process algebra, *in* ‘Proc. 15th Twente Workshop on Language Technology’.

Constructing Real-Time Collaborative Software Engineering Tools Using CAISE, an Architecture for Supporting Tool Development

Carl Cook **Neville Churcher**

Software Engineering & Visualistion Group, Department of Computer Science & Software Engineering,
University of Canterbury, Private Bag 4800, Christchurch, New Zealand
E-mail: {carl,neville}@cosc.canterbury.ac.nz

Abstract

Real-time Collaborative Software Engineering (CSE) tools have many perceived benefits including increased programmer communication and faster resolution of development conflicts. Demand and support for such tools is rapidly increasing, but the cost of developing such tools is prohibitively expensive. We have developed an architecture, CAISE, to support the rapid development of CSE tools. It is envisaged that the architecture will facilitate the creation of a range of tools, allowing the perceived benefits of collaboration to be fully realised. In this paper, we focus on the development of CSE tools within the CAISE architecture. We present examples to illustrate how such tools are constructed and how they support real-time multi-user collaborative software development. We also address issues related to the number of collaborators and discuss performance aspects.

Keywords: Collaborative Software Engineering, CSCW & Groupware, Tool Construction, Continuous Integration

1 Introduction

Software engineering is a predominantly collaborative activity. Typically multiple teams of people develop and maintain successive versions of a range of products, in parallel. Surprisingly, tools to support synchronous, or real-time Collaborative Software Engineering (CSE) are still restricted to minor tasks for specific software engineering purposes—if they make it out of the research prototype stage at all.

Today there is a real need for CSE tools, and this demand will grow as software engineering becomes an increasingly complex and heterogeneous discipline. While support for collaboration has emerged in other areas of everyday applications such as file sharing, instant messaging, and generic tele-working, software engineers themselves appear ambivalent about the opportunities and potential benefits of more comprehensive tool support. Accordingly, research into CSE is both timely and imperative.

The main premise of our research is that by enabling fine-grained CSE through seamlessly integrated tool support, it is possible to raise the very restricted levels of communication within current software engineering practice. The value of active communication has long been recognised in Computer Supported Collaborative Work (CSCW) research (Greenberg 1989), and is re-emerging in software engineering within the eXtreme Programming movement. Increased programmer communication, as put forward current CSE research, is likely to produce more informed decisions during the development stage of software engineering, and less likelihood of costly coding conflicts.

In this paper, we discuss how new tools can be constructed within the CAISE architecture to support the real-time development of a collaborative software project. We also address issues related to large group sizes and performance aspects of the architecture. The design of the CAISE architecture has been described previously (Cook, Churcher & Irwin 2004).

The remainder of the paper is structured as follows. Section 2 provides a background related to real-time CSE and supporting tools. Section 3 provides an overview of the CAISE architecture with particular emphasis on the services it provides to the construction of new CSE tools. Section 4 presents some typical CAISE-based CSE tools in use today, and Section 5 illustrates how to build new CSE tools within the CAISE architecture. Section 6 addresses the performance of the CAISE architecture and supporting tools, and a summary is given in Section 7.

2 Background and Motivation

In the last year many of the major commercial IDEs have taken significant steps towards code-level real-time collaboration. Of the five Java IDEs that have the largest market shares, Eclipse, Borland's JBuilder and Sun's JSE now support shared development facilities, and all vendors are promising more to come in the next major releases.

While the proposal of tools to support CSE often draws an enthusiastic response from practitioners, the design and implementation of commercial-strength tools is a challenging task. Even once such tools have been developed, there is no guarantee that they will gain widespread adoption; this mistake has been made within related areas of CSCW research (Carasik & Grantham 1988).

Any CSE tool has complex issues to address, such as user interface design, CSCW floor control and management, varying levels of collaboration requirements, varying expectations between developers within a group, support of multiple artifact types, and potentially multiple views of artifacts. There are also technical aspects to address such as concurrency control and distributed system design, along with the standard software engineering technicalities such as parsing, semantic modelling and source code management.

Very few research prototypes have evolved into features within professional tools. A significant difficulty is that conventional software engineering tools are designed for single-developer use, and 'bolting on' collaborative features does not necessarily scale or provide the level of improvement envisaged.

Implementing collaborative features for commercial strength tools, as we and others have discovered, is a very challenging problem. To date, software engineering tools typically work with the lowest common denominator of software engineering artifacts: source code. By employing source files as the finest-grained type of shared information, and by supporting information sharing only through code repository systems, it is difficult to extend IDEs to support real-time within-files collaboration, to provide support for additional views of software, and to provide collaborative access to the underlying software model.

While it is certainly possible to implement collaboration-enhanced real-time software engineering tools, the single significant barrier to the success of tools may be the poor ratio of tool power to development effort.

2.1 Why CSCW Fails for CSE

An initial solution to implementing CSE tools is to use CSCW Groupware, and this has been trialled elsewhere with varying degrees of success (Schummer 2001) and (Churcher & Cerecke 1996). Groupware toolkits such as GroupKit (Roseman & Greenberg 1996) and Maui (Hill & Gutwin 2003) allow the sharing of text documents, whiteboards and other common forms of electronic media, and good results have been achieved when converting single user generic applications to their multi-user equivalents (Cox & Greenberg 2000).

Problems occur, however, when building industrial-strength CSE tools from Groupware toolkits. Professional tools are not limited to a single task, a single language or a single artifact view, and this is orthogonal to the characteristics of Groupware support. Additionally, Groupware has no understanding of the complex semantics or syntax of artifacts, and the relationships between users and artifacts. Software engineering artifacts are also persistent over a long time scale, whereas Groupware is more aligned to the support of transient communication with throw-away artifacts.

An attempt could be made to extend a single-user IDE collaboratively through the use of a CSCW toolkit, allowing it to support distributed collaborative development of code or UML diagrams. However, extending it where multiple views of artifacts are supported at the same time, such as round-trip engineering between source code and UML diagrams, is extremely difficult. This is particularly true if the main source of information is derived from the source code management system; shared access to an in-depth semantic model of the project is typically required to translate between different views. Commercial IDEs do not provide this functionality adequately, and IDE model sharing is only in its infancy within the Eclipse platform via the Eclipse Communication Framework project (Lewis 2005).

Given that CSCW technology does not scale to meet the needs of CSE, and IDEs do not provide enough fine-grained information to support the development of highly-synchronous new tools, often the only means of producing new tool sets is by completely redesigning tools upon a foundation of collaborative software engineering technology.

2.2 Related Work Towards Real-Time CSE

The augmentation of software engineering tools with Groupware features failed to produce tools of any considerable impact within the field of research. A decade on since the conception of Groupware toolkits, researchers have turned to implementing collaboration-based prototype tools upon existing software engineering frameworks.

Palantír, for example, is a system that inspects in real-time the impact of changes within source code repositories (Sarma & van der Hoek 2002). Rosetta provides an Internet-based collaborative class diagramming tool-set (Graham, Stewart, Ryman, Kopaee & Rasouli 1999). Tukan, using the COAST Groupware framework, is a shared code editor for SmallTalk (Schummer 2001). Augur provides comprehensive user activity visualisations based on source code management systems (Froehlich & Dourish 2004). Moomba is a recently published system for supporting distributed eXtreme Programming (XP) within a global software development context (Reeves & Zhu 2004).

Tools such as those listed above are well suited for a single task or development methodology, but they are for the most part fixed and non-extensible. Additionally, it is certain that each tool took a considerable amount of effort to design and implement. The purpose of our research is to reduce the barrier of high construction costs for CSE tools by proving a architecture that enables many types of CSE tools to be developed rapidly.

3 The CAISE Architecture

The CAISE architecture, as illustrated in Figure 1, allows isolated programmers to work collaboratively without sacrificing communication. CAISE-based tools achieve this by keeping all programmers within a group synchronised in real-time, at the same time providing customisable user awareness and project state information to the individual tools.

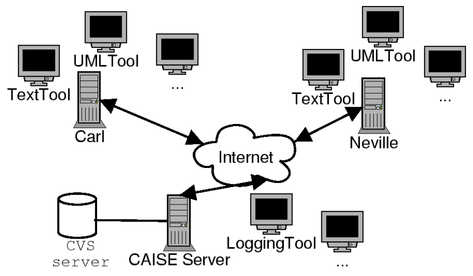


Figure 1: A schematic representation of CAISE.

The CAISE architecture provides an infrastructure with the potential to support the entire software engineering process. CAISE tools can be constructed that provide more that just the shared editing of basic software artifacts. Collaborative compilation, testing and debugging of software projects are also possible to implement using the services of CAISE. Comprehensive inter-developer communication facilities can also be constructed.

The CAISE architecture is not built on top of a source code repository system. CAISE and its supporting tools, however, do not aim to replace source code repositories either. The ability to work in private at times and to be able to keep different versions of programs separate are elements that very few programmers could do without. Our tools, therefore, are designed to support what code repositories do not provide: communication between developers and tools during fine-grained real-time collaboration, such as multi-user coding within the same file.

The example CSE tools presented in this paper support the core functionality expected of any software engineering suite, including project compilation and execution support, editor undo, cut, copy and paste, UML class diagramming, and round-trip engineering between tools.

To date, the current set of CSE tools have performed well anecdotally (Cook et al. 2004), empirically (Cook & Churcher 2005*a*), and heuristically (Cook & Churcher 2005*b*). The underlying CAISE architecture allows for the rapid development of fully featured CSE tools, such as those presented in this paper. Typically, CAISE tools are designed to support patterns of collaboration evident in software engineering practice. Such patterns are presented in (Cook 2005).

3.1 Comparison to Other CSE Architectures

We are aware that different programmers will have different tool requirements, and we make no assumptions about the 'right' set of software engineering tools. Accordingly, CAISE has been implemented as an extensible architecture rather than a tool-set. We have focused on designing a architecture that can support a wide range of custom collaborative applications.

In contrast to most other tools, CAISE employs a holistic approach in that the entire infrastructure is based upon collaboration. At the core of the architecture lies a shared semantic model of the software being developed, allowing multiple views of artifacts to be supported by any number of different tools. The CAISE server, which houses each project's semantic model, exists as a shared IDE engine for collaborative tools. This is depicted in Figure 1, where all management, parsing and semantic analysis of shared artifacts is provided by default.

The semantic model housed within the CAISE server is primarily focused on object oriented languages. Full support is available for Java 1.4, with work near completion for Java 1.5, including Generic Types. Other 'Java-like' languages such as C$^{\#}$ can be supported by inserting a new semantic analyser into the CAISE server. Such an analyser maps C$^{\#}$ parse trees into a model-based program structure, which the CAISE server integrates into the project's semantic model.

CAISE is not simply a collaborative add-on project to existing single-user tools. CAISE operates as a fine-grained Continuous Integration (CI) platform where all tools are synchronised with each other, as opposed to systems that periodically attempt to resynchronise through source code repositories and social protocols. The CAISE architecture also differs by being capable of supporting complex CSE tools. It is not restricted to particular tasks or methodologies—there is no theoretical limit to the scale and ability of CAISE-based tools.

3.2 Tool Services

CAISE provides services which support the rapid development of CSE tools. By utilising the CAISE architecture, CSE tools can rely on the CAISE server to manage the storage and sharing of artifacts, and to control users as they join and leave projects and artifacts. CAISE also provides the low-level mechanisms to allow distributed messaging between tools and the CAISE server, and supports a distributed event model.

A semantic model of the software for each project is maintained by the server, which is refined upon the actions of participating CAISE tools. The semantic model represents the collection of components within the software project, such as packages, classes, methods, properties, and relationships such as inheritance, association and invocation. CAISE-based tools are not required to perform any parsing or semantic analysis themselves; the server is responsible for translating modifications in artifacts to an updated semantic model. The model, however, is accessible by CAISE tools both for reading and direct modification if required.

The functions provided by the CAISE server, both in terms of supporting collaborative work and performing core software engineering tasks, allow the CSE tool developer to focus on the specific requirements of the given tool rather than re-implementing the functionality common to most CSE tools. If, however, the tool being developed requires additional features, the CAISE architecture is easily extended to accommodate new artifact types and kinds of feedback. Section 3.7 discusses this concept further.

3.3 CAISE Tool Widgets

Before we look at the construction of individual CAISE-based tools, this section introduces some of the collaborative widgets available for use within any CSE tool. The following widgets can be added to Swing/AWT-based Java applications without any additional coding requirements, which allows tools to be augmented with CSE capabilities for minimal effort. These widgets operate by communicating with the CAISE server and responding to real-time events.

The *User Tree* is shown in Figure 2, which may be used within a tool to support user awareness. This widget provides a user-centered view of the CAISE-based software engineering project in real-time. Individual tools require no knowledge of the artifacts they are editing; they simply have to keep the CAISE server informed of the name of the artifact currently being edited and the most recent cursor location of the user controlling the tool. The User Tree will keep itself updated with the latest view.

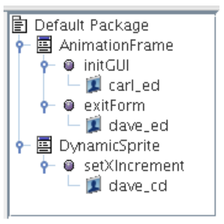


Figure 2: The CAISE *User Tree* widget, supporting a user-centric project view.

The *Change Graph* is another widget that can be readily added to any CSE tool. This widget is illustrated in Figure 3. The Change Graph widget keeps track of the cumulative additions and deletions to and from the model on a per-user basis. This provides each tool user with an overview of the current development activity. Again, this component can be added to any CSE application, or housed in a dashboard display or separate frame.

The *Client Panel* is key component of the CAISE widgets package, and can be seen within the CAISE-based tool presented in Figure 7. The Client Panel typically houses four components known as the *Artifacts*, *Users*, *Feedback* and *Build Panes*, although the Client Panel can be configured to house any combination of specific panes. Individual panes can also be added to an application separately.

The Artifacts Pane is presented at the bottom of Figure 7. This provides basic file information on the artifacts within a CAISE project, including their current compilation state. The Users Pane is presented in Figure 5. This pane allows messages to be sent between users, including audio broadcasts.

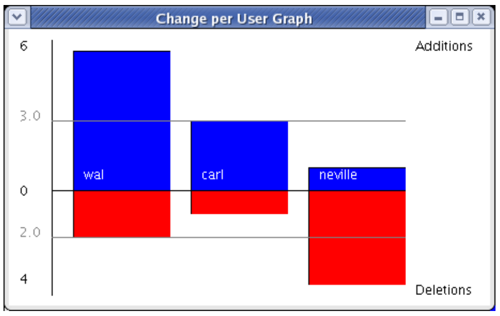
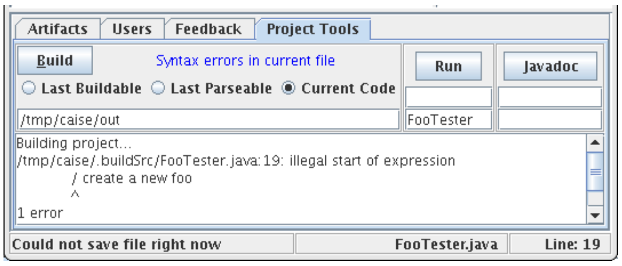


Figure 3: The project *Change Graph* widget.

The Build Pane is presented in Figure 4. It provides an adjustable level of collaborative awareness, allowing the user to temporarily ignore concurrent edits for the purpose of building the system without interruption. In addition to allowing the project to be built from the live, last parseable or last buildable version, the Build Pane also allows the current project to be executed. Finally, the Feedback Pane, which is not presented in this paper, displays server-based plain-text information such as Degree of Awareness (DOI) feedback between users and impact reports that result from artifact modifications.



Syntax errors in current file

/tmp/caise/out

FooTester

```
Building project...
/tmp/caise/.buildSrc/FooTester.java:19: illegal start of expression
        / create a new foo
        ^
1 error
```

Could not save file right now | FooTester.java | Line: 19

Figure 4: The CAISE *Build Pane* with adjustable levels of collaborative awareness.

The CAISE Tool widgets can be utilised from within any application, and applications that use such components do not require any specific software engineering knowledge or capabilities. For example, a stand-alone text editor can be enhanced by incorporating the CAISE collaborative User Tree into its user interface. Given that the text editor conforms to the CAISE Tool Protocol as specified in Section 3.5, the User Tree will highlight the method, class and package that the editor is currently modifying, without the editor needing to possess any specific software engineering capabilities.

Within the next release of CAISE, we also aim to add the multi-user text pane to our collaborative widgets package. The text pane is illustrated in Figure 7.

3.4 The CAISE Tool API

The *CAISE Tool API* (CTA) is provided as the means of accessing the functions of the CAISE server from within a CSE application. While the CAISE server typically resides on a separate machine, the CTA allows the calling application to view the server as if it was contained within the same process; the server functions appear no different to those of any other library. The server is accessed by a set of standard method calls, data is marshalled as method return values, and catchable events are thrown whenever interesting actions occur during the development of a CAISE project.

Table 1 presents several of the key CTA methods. This is only a subset of the complete CTA, but it provides a useful overview of the programming interface.

The CTA provides adequate functionality to implement numerous types of CSE tools. Multi-user text editors, for example, can rely on the CTA to provide collaborative code editing, semantic analysis of code modifications, and user presence feedback. To implement communication facilities, messaging can be provided via the Chat methods, and audio broadcasts are also supported. Tool design and implementation will always be the responsibility of the CSE researcher, but the CTA prevents ‘reinventing the wheel’ for the essential yet complex CSE services.

CAISE is a concurrent system, where it is likely to receive multiple interleaved requests to modify a set of artifacts within a short period of time. Invocations of CTA methods are treated fairly at the CAISE server. In the underlying distributed system that CAISE employs for its client interface, each incoming method call is queued and then processed in a thread safe, sequential order. For all other pending method calls in the queue, a low-CPU blocking mechanism is used on the client side.

3.5 The CAISE Tool Protocol

By following the *CAISE Tool Protocol*, which specifies the contract between individual tools and the server, tools are assured of staying synchronised with each other, and the server is able to avoid concurrency issues such as deadlocks and forced rollbacks of tool requests.

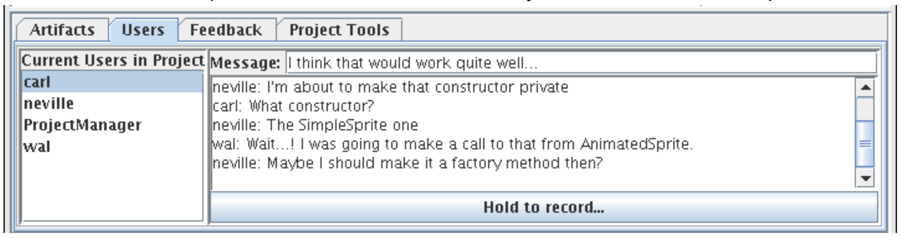
Individual CSE tools have the ability to implement locks and other floor control policies that allow only one user at a time to edit a given region of code. By default, however, the CAISE architecture allows full synchronous editing of any artifact. To ensure that tools are always synchronised, a specialised Model-View-Controller (MVC) approach is used which guarantees consistency over distributed parallel edits. Requests to edit the view are captured by tools, but the view is not immediately updated. Rather, the edit is sent to the server which in turn edits the global model, and broadcasts the change to all tools. Each tool then updates its local view, including the tool that made the edit request.

To implement a CSE tool that adheres to the CAISE Tool Protocol, three application-level threads are typically used: a GUI thread, a worker thread, and a CAISE event listener thread. The threading model for CAISE-based tools is presented in Figure 6. Most windowing toolkit libraries provide a GUI thread, and the CAISE Tool API provides a CAISE event listener thread. As the worker thread can simply be the main application thread, it is unlikely that any new threads need to be created explicitly within a CAISE tool. With the existence of a worker thread, the GUI thread is free to take any volume of user input from the user interface, without causing jitter or lag as the events are being proxied to the CAISE server.

By using a MVC approach and following the CAISE Tool Protocol, CSE tools are guaranteed to stay up-to-date and synchronised with the CAISE server, and there is no risk of deadlocks or loss of information. The following list presents the six key conditions of the CAISE Tool Protocol:

1. The CSE tool captures all user input events such as keystrokes and caret move events, typically using action listeners. All actions are to be consumed, blocking the underlying view of the artifact from being modified.

2. All captured events are placed into a FIFO event queue within the CSE tool. The GUI thread returns immediately after placing the event in the



I think that would work quite well...

neville: I'm about to make that constructor private
carl: What constructor?
neville: The SimpleSprite one
wal: Wait...! I was going to make a call to that from AnimatedSprite.
neville: Maybe I should make it a factory method then?

Figure 5: The CAISE *Users Pane*, which provides voice and text communication.

Method	Description
Connect to Engine	Makes a new connection to the given CAISE server
Open Project	Opens an existing CAISE project
Add Artifact	Adds a new artifact to the given project
Open Artifact	Sets an existing artifact as open for a given user
Set User Location	Moves a user's cursor location within an artifact
Update Source Code	Appends a sequence of characters to an artifact
Update Parse Tree	Appends a parse tree of an artifact
Update Model	Directly manipulates the semantic model of a project
Get Model Snapshot	Returns a copy of a project's semantic model
Fire Tool Event	Allows a tool to invoke tool-specific server plug-ins
Get Event Log	Returns the complete event log for a given project
Send Chat Message	Allows users to send text messages between tools

Table 1: Key methods of the CAISE Tool API.

queue, preventing any latency within the user interface.

3. A separate CSE tool worker thread dequeues events and issues them to the server as corresponding CTA method invocations.

4. The CSE tool worker thread waits for the return value of the CTA method invocation before processing the next tool input event. The CSE tool does nothing upon a successful method invocation, and escalates any errors if the method invocation fails.

5. The CSE tool's CAISE event listener thread listens for broadcasted server events that result from CTA method invocations. Upon relevant events such as artifact modifications and user location changes, the model of the artifact within the tool is updated accordingly. This step is performed by all participating tools, not just the instance that invoked the event.

6. Upon any model update, the CSE tool's artifact view is redrawn by the GUI thread.

During spikes of development by multiple CAISE tools, the server ensures fairness by queuing events evenly based on the inbound tool connection, rather than order of arrival. In this manner, we avoid the situation where all other tools are unfairly delayed by an exceptionally active single user.

3.6 Tool Events

Within the CAISE event model, the CAISE server broadcasts events of various types to all participating tools upon actions of individual users; this is illustrated in Figure 6. Events contain details of the general action, such as an artifact edit by a named user, and the specific details such as the affected text and file offset.

Each CAISE event type is briefly summarised in Table 2. Fully featured tools are likely to register as a listener for all events, as can be seen in the code listing in Section 5.1. Other components, such as the Change Graph presented in Section 3.3, are only interested in specific event types.

Type	Typical actions
Project	A project is created or deleted.
Artifact	An artifact is added, removed or edited.
Chat	A user issues text or audio messages.
Client	A client opens, closes or moves location within an artifact, or rebuilds a project.
Change	The project model is manipulated directly or via artifact modification.
Plug-in	Tool-specific custom units of information exchange.

Table 2: Events types within the CAISE architecture.

3.7 Tool Manager Modules

The CAISE architecture provides generic support for the collaborative editing of text documents, the parsing of source files, and the semantic analysis of parse trees derived from source code and UML diagrams. Often, however, tools require further functionality from the server, including the support of new artifact types. To accommodate extensibility within the CAISE server, modules known as *Tool Managers* can be integrated through the CAISE plug-ins interface.

For a UML class diagrammer, it is apparent that such a tool requires information beyond what is contained within the core semantic model of the project. As well as displaying all classes, methods and relationships, a class diagrammer contains class layout information that must be shared every time any instance of the class diagram is modified. Therefore, when implementing the UML diagramming tool presented in this paper, an additional diagrammer-specific type of artifact was introduced, managed and shared by a UML-specific Tool Manager.

For the UML diagramming tool, whenever a user changes the location of a displayed class, a tool-specific event is thrown to the CAISE server via the CTA and this event is proxied to the UML diagrammer Tool Manager. The Tool Manager will then access and update the new artifact that stores the class location information, and then broadcast this change out to all tools in accordance with the CAISE Tool Protocol, allowing all users to update their local view of the UML class diagram.

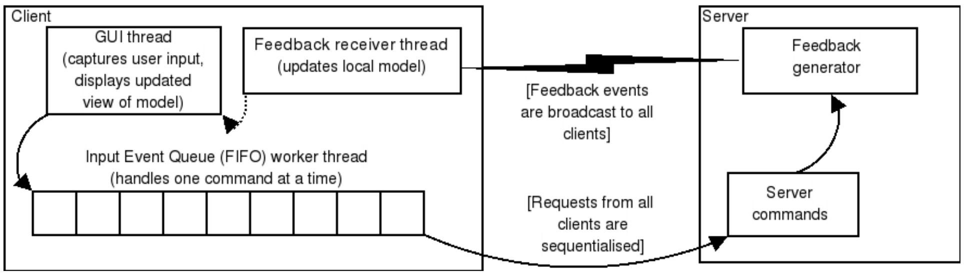
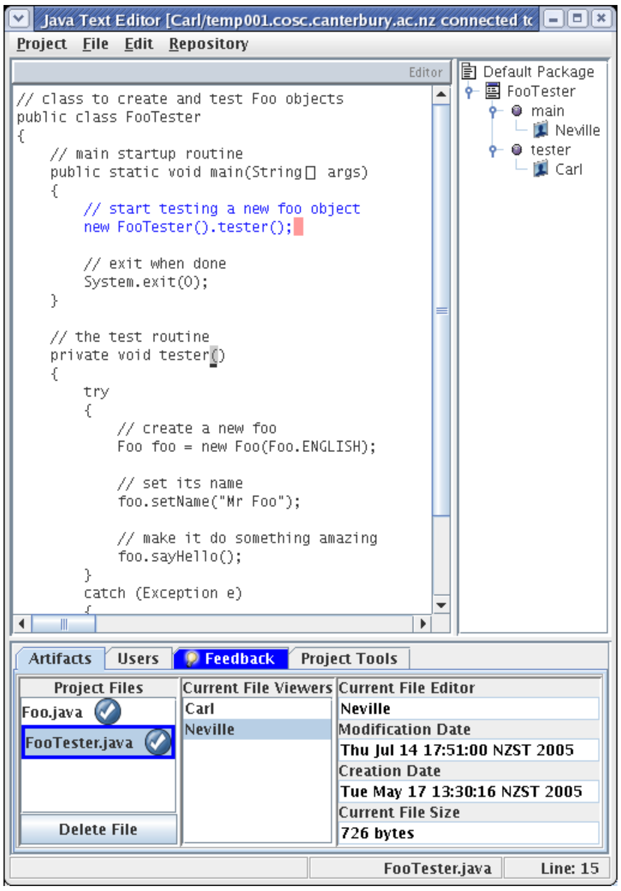

Client
GUI thread (captures user input, displays updated view of model)
Feedback receiver thread (updates local model)
Input Event Queue (FIFO) worker thread (handles one command at a time)
[Feedback events are broadcast to all clients]
[Requests from all clients are sequentialised]
Server
Feedback generator
Server commands

Figure 6: The recommended threading model within a CAISE-based tool.

The CAISE server can be extended in many other ways through the plug-ins interface, including support for new languages. For further details please refer to (Cook et al. 2004).

4 Example CAISE-Based Tools


Java Text Editor [Carl/temp001.cosc.canterbury.ac.nz connected tc
Project File Edit Repository
Editor
Default Package
FooTester
main
Neville
tester
Carl
Artifacts Users Feedback Project Tools
Delete File
FooTester.java
Line: 15

```
// class to create and test Foo objects
public class FooTester
{
    // main startup routine
    public static void main(String[] args)
    {
        // start testing a new foo object
        new FooTester().tester();

        // exit when done
        System.exit(0);
    }

    // the test routine
    private void tester()
    {
        try
        {
            // create a new foo
            Foo foo = new Foo(Foo.ENGLISH);

            // set its name
            foo.setName("Mr Foo");

            // make it do something amazing
            foo.sayHello();
        }
        catch (Exception e)
        {
```

Project Files	Current File Viewers	Current File Editor
Foo.java ✓	Carl	Neville
FooTester.java ✓	Neville	Modification Date
		Thu Jul 14 17:51:00 NZST 2005
		Creation Date
		Tue May 17 13:30:16 NZST 2005
		Current File Size
		726 bytes

Figure 7: A CSE code editor. When viewed in colour the text highlighting and tele-cursors are visible.

Since their conception, we have been constantly refining the CSE tools discussed in this paper in order to provide realistic Software Engineering environments. These tools are presented in Figures 7 and 8. The main features currently supported include:

- Round-trip engineering between all tools and artifact views.
- Full multi-user editing and UML class diagramming capabilities with a *relaxed WYSIWIS* view, including collaborative undo.
- An Artifacts Pane that displays the current compilation state of each artifact as well as editor details and file information.
- A multi-user text pane which provides remote modification highlighting and *tele-cursors*. The diagrammer pane also indicates remote developer locations through special markers and tool-tips.
- Instant messaging and an audio chat channel.
- All relevant aspects of the user interface have been designed to accommodate the constantly changing state of each developer's display.
- A source code control system has been integrated to allow a CAISE project to access a central code repository.
- Build and run facilities, including protection from crosstalk when attempting to compile during times of high development activity
- Event-based collaborative feedback information, such as Degree of Interest (DOI) reports relating to other user locations within the project, and model change impact reports as the project evolves.
- A collaborative User Tree that provides a model-based view of developer's locations.

The majority of the features built in to the above tools, such as the Artifacts Pane and User Tree, are stand-alone components made available from the CAISE client widgets library, as presented in Section 3.3. The remaining collaborative features, such as multi-user editors and a UML class diagramming pane, have been implemented manually, but rely on the services of the CAISE Tool API to implement functions such as tool synchronisation and the shared modification of artifacts.

To implement tele-cursors within the Java code editor, each instance of the editor simply listens for changes in remote user cursor locations and redraws each remote cursor onto the text pane using opaque graphics rendering. To provide more advanced Groupware features, components of the Maui toolkit could be integrated with any CAISE-based CSE tool.

To implement a relaxed-WYSIWIS view within the UML class diagrammer, each time a component such as a class or interface is dragged, the drag action is captured and sent to the CAISE server via the `fireToolEvent()` API method. The UML diagrammer Tool Manager module responds to this event by adjusting its mapping of components and coordinates, and then broadcasts the adjustment event out to all tools registered for this event, which in turn update their local view of the model.

There are numerous other features that have been built into other tools such as code-age highlighting, as well as stand alone graphical components such as real-time visualisations of user activity. These features have been presented previously (Cook et al. 2004) (Cook & Churcher 2003), where the primarily focus was on describing the CAISE infrastructure.

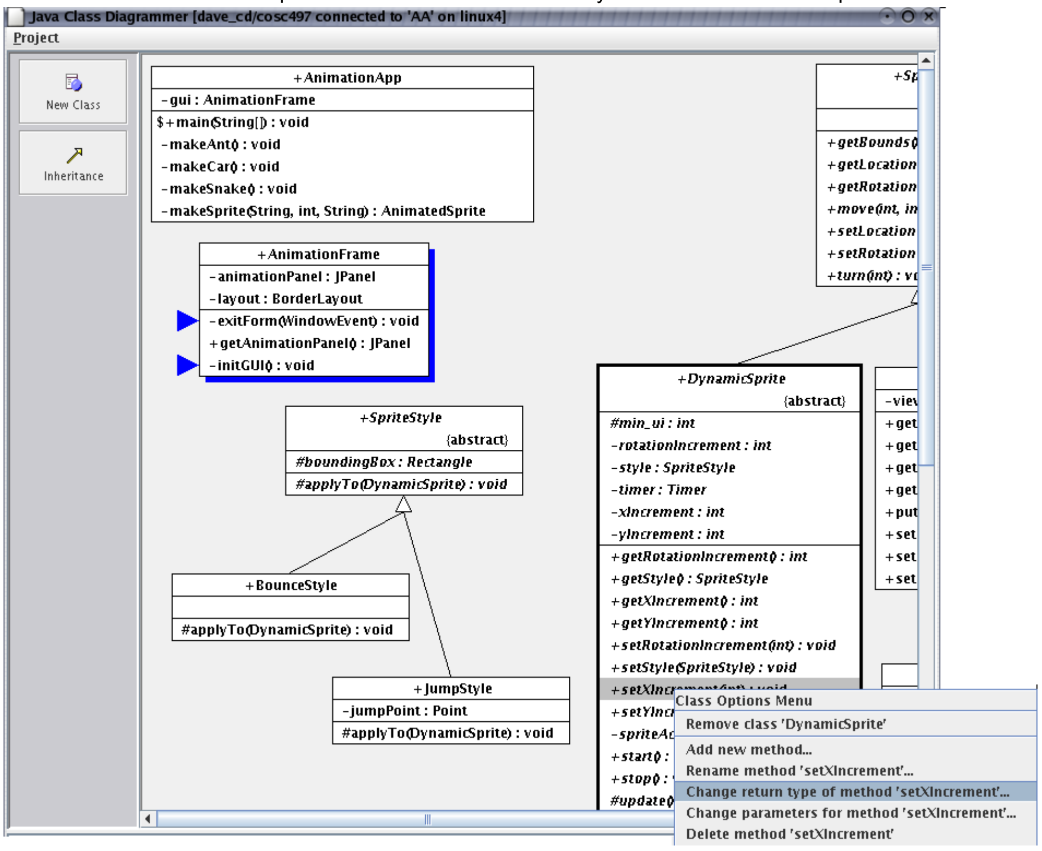
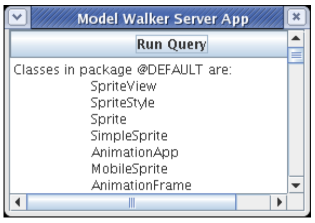

Java Class Diagrammer [dave_cd/cosc497 connected to 'AA' on linux4]
Project
New Class
Inheritance
+AnimationApp
-gui : AnimationFrame
$+main(String[]) : void
-makeAnt() : void
-makeCar() : void
-makeSnake() : void
-makeSprite(String, int, String) : AnimatedSprite
+AnimationFrame
-animationPanel : JPanel
-layout : BorderLayout
-exitForm(WindowEvent) : void
+getAnimationPanel() : JPanel
-initGUI() : void
+SpriteStyle
{abstract}
#boundingBox : Rectangle
#applyTo(DynamicSprite) : void
+BounceStyle
#applyTo(DynamicSprite) : void
+JumpStyle
-jumpPoint : Point
#applyTo(DynamicSprite) : void
+DynamicSprite
{abstract}
#min_ui : int
-rotationIncrement : int
-style : SpriteStyle
-timer : Timer
-xIncrement : int
-yIncrement : int
+getRotationIncrement() : int
+getStyle() : SpriteStyle
+getXIncrement() : int
+getYIncrement() : int
+setRotationIncrement(int) : void
+setStyle(SpriteStyle) : void
Class Options Menu
Remove class 'DynamicSprite'
Add new method...
Rename method 'setXIncrement'...
Change return type of method 'setXIncrement'...
Change parameters for method 'setXIncrement'...
Delete method 'setXIncrement'

Figure 8: A UML class diagrammer. Remote user positions are indicated by the blue markers.

4.1 Server Applications

The most common type of CAISE-based tools are those that allow the direct editing, building and inspection of collaborative software projects. Other more static types of tools can be envisaged, however, such as visualisation generators and metrics querying tools. For this class of tool, the CAISE architecture supports *server-based* applications. These applications run within the server process itself, providing fast and efficient access to the software project, its artifacts and underlying semantic model.



```
Classes in package @DEFAULT are:
        SpriteView
        SpriteStyle
        Sprite
        SimpleSprite
        AnimationApp
        MobileSprite
        AnimationFrame
```

Figure 9: A server application which inspects the semantic model of a CAISE-based software project.

An example of a simple server application is presented in Figure 9. This is a trivial example, where the application simply walks a project model and displays the names of all classes within the package structure. The following code segment, however, shows how simple it is for the above application to walk the model programmatically through the use of the architecture's *model visitor*. Server applications can modify the model directly as well, causing model change events to be issued to all listening tools, which will in turn adjust their local views accordingly.

```
public class SimpleModelWalker extends CAISEServerApp {

    // called once server is ready
    public void run() { initGui(); }

    // called upon events
    public void update(Collection events) { /* do nothing */ }

    /* InitGui() method omitted */

    public void jButton1ActionPerformed(ActionEvent e) {

        // get the given project
        Project project = Engine.getEngine().getProject("AA");

        // get the default package from the project's model
        PackageDecl pkg = project.getModel().getDefaultPackage();

        // print out header
        setText("Classes in package " + pkg.getSimpleName());

        // create an instance of a subclassed model visitor
        new ModelVisitorAdapter(pkg) {

            // override the visit ClassType routine
            public void visitClass(ClassType classType) {

                // write the class name out to the text panel
                addText("\n\t" + classType.getSimpleName());
            }
        }.visit(); // fire up the adapter
    }
}
```

5 Construction of New Tools

This section describes how the tool developer can construct new CSE tools using the CAISE framework. The code segments presented in this section are taken from the Java code editor, which was discussed in the previous section and presented in Figure 7.

5.1 Tool Initialisation

Each instance of a CAISE-based CSE tool needs to establish a connection to the CAISE server. The most appropriate time to do this is at program startup. Within CAISE, each client has a unique name, and this is given during the call to establish a server connection. The following code segment demonstrates connecting to a named CAISE server, registering an application for CAISE events, and opening an existing project.

```
// create a new instance of a CAISE handler
CAISEHandler handler = new CAISEHandler(clientName,
                                        serverName, TextEditor.ID);

// tell the server to notify this class of any events
handler.attachCAISECallback(this);

// open the initial CAISE project with no event filtering
handler.openProject(projectName, CAISEEvent.ALL_EVENTS);
```

5.1.1 Adding CAISE Widgets to a Tool

CAISE widgets may be constructed and added as components within a user interface in the same manner as any other standard graphics widget. The only requirement is that events from the CAISE server are passed from the containing application to each widget. The event handling code segment is given in Section 5.4.

5.2 Catching Local Tool Actions

The CAISE Tool Protocol stipulates that tools pass artifact modification events to the CAISE server, instead of allowing the tool's view of an artifact to be modified directly. To do so, tools must catch all artifact modification actions and queue them for subsequent proxying to the server. Only once the event has been processed by the server and a response has been broadcasted to all tools will the local text pane be updated, as illustrated in Section 5.5.

In the following code segment, the key-presses destined for the text pane within the Java editor are captured and queued. The current cursor location is not recorded within the keystroke event—the server maintains the authoritative record of user positions to ensure consistency between all the tools, and already knows the user location at the time of the pending key press. To maintain the record of user locations, cursor location changes are another type of CAISE event governed by the CAISE Tool Protocol.

```
public void keyTyped(KeyEvent e) {

    // kill it before it gets to the editor
    e.consume();

    // ignore any keystrokes that involve alt or ctrl
    if ( (e.getModifiers() & (e.ALT_MASK|e.CTRL_MASK)) )
        return;

    // ignore escape key
    if (e.getKeyChar() == (char)27)
        return;

    // add regular key event to client queue
    enqueEvent(new EventWrapper(e, fileName));

    // update the state of the undo menu item
    EditorFrame.this.undoItem.setEnabled(true);
}
```

5.3 Sending Tool Actions to the Server

As described in Section 3.5, the GUI thread is only responsible for capturing and enqueuing user input, and updating the local view of artifacts. The role of the worker thread is to take events from the local event queue and deliver them to the server as API method calls. As illustrated in the following code segment, the worker thread blocks until a corresponding event has been broadcast by the server before processing any remaining queued events.

```
final class EventHandler implements Runnable {

    public void run() {
        while (isThreadRunning()) {

            // remove event from queue and pass to server
            EventWrapper ew = clientInputEvents.take();

            if (ew.event instanceof KeyEvent)
                // send key events as buffer append requests
                handler.appendSourceCodeBuffer(ew.fileName,
                                               ew.event);

            // wait until the server has replied
            serverFeedbackEvents.take();
        }
    }
}
```

5.4 Listening for Server Responses

The CAISE server broadcasts events out to all registered listeners upon any significant event such as an artifact modification or a change in the project's underlying semantic model. If a tool has issued a request to modify an artifact, the server will perform the modification on its master copy and then broadcast a corresponding event to all tools. The tool that issued the request will be expecting a subsequent modification event, and all other tools are also required to adjust their local artifact views upon event notification.

The following code segment illustrates the main event loop within the Java text editor, which is representative of typical CAISE-based tools. As the editor also employs the User Tree widget, events are relayed to the widget, allowing it to update its own view of the project. The text editor also needs to keep track of user location changes in the same manner as it monitors artifact modification events, but for the sake of simplicity, this has been omitted from this example.

```
public void update(Collection events) {

    // for each event
    for (Iterator i = events.iterator(); i.hasNext(); ) {
        CAISEEvent event = (CAISEEvent)i.next();

        // inspect the event type
        switch (event.getType()) {

            // if an artifact event
        case CAISEEvent.ARTIFACT_EVENT:

            // if the artifact has been edited by anyone
            if (event.getSubType() == ARTIFACT_APPENDED)

                // if this is the current artifact
                if (event.getSourceEntity().equals(fileName))

                    // append the buffer of the underlying file
                    appendBufferFromRemoteChange(
                      event.getSourceUser(),
                      ((KeyEvent)(event.getData())[0]),
                      ((Integer)(event.getData())[1]).intValue(),
                      ((Integer)(event.getData())[2]).intValue());
        }

        // update user tree
        userTree.updateTree(event);
    }
}
```

5.5 Updating the Local Artifact View

To complete the MVC pattern within the CAISE event model, the final task for CSE tools upon receiving an event is to update their local view. Within the text editor, this involves appending and redisplaying the text pane upon artifact modification events.

For user location change events, this involves updating the local mapping of users and file positions and redisplaying all cursors. As the Java code editor uses a multi-user text component, artifact modification events only need to be relayed to the text pane—the multi-user component will perform the text insertion and remote modification highlighting internally.

It is important to note that each tool's view runs no possibility of losing synchronisation with other tools or the CAISE server, baring catastrophic network failure. As long as events are captured and delivered in order to the server, and the underlying artifact is only updated within each tool in response to server events, then synchronisation is guaranteed.

```
private void appendBufferFromRemoteChange(Client editor,
                                          KeyEvent change,
                                          int positionHint,
                                          int previousFileSize) {

    // check that our user location is in sync with the server
    assertUserLocation(editor, positionHint);

    // check that the file size is in sync with the server
    assertFileSize(previousFileSize);

    // update buffer
    buffer.appendDocument(change.getKeyChar(), positionHint,
                          editor, handler.getClient());

    // restore any previously selected text
    redrawSelection(editor.equals(handler.getClient()));

    // tell auto-save timer to restart
    setBufferDirty(true);

    // yeild lock if this edit originated from this app
    if (editor.equals(handler.getClient()))
        serverFeedbackEvents.put(new Object());
}
```

6 Performance Analysis

A final consideration when discussing the design and use of collaborative tools for software engineering is that of performance. The performance of the tools must be satisfactory, and there should be no theoretical limitations of the architecture that will prevent the tools from being useful in realistic environments. While the core response speeds and resource usage of CAISE and its supporting tools have proved acceptable over a long period of subjective testing and user evaluations, it is important to note the effects of code size and number of concurrent developers on server memory load and tool response times.

6.1 Memory Load

To provide features such as code modification impact reports and DOI feedback, the CAISE server maintains a semantic model of the software within the project. An immediate concern is that of memory usage; if a large amount of memory is required for each line of code added to the model, projects of a realistically large size might be beyond the scope of the CAISE architecture.

Figure 10 presents the amount of memory used per line of code across a range of CAISE projects. For any CAISE based project, the server first loads in all packages, classes, interfaces and methods directly accessible from any Java source file. This brings the initial project model size up to around 60 MB. From that point onwards, however, most of the components that the modelled software rely upon are now loaded, and the project model size increases only linearly in relation to the number of classes and methods declared in each source file. After taking the project initialisation into account, each line of code requires approximately one kilobyte of server memory.

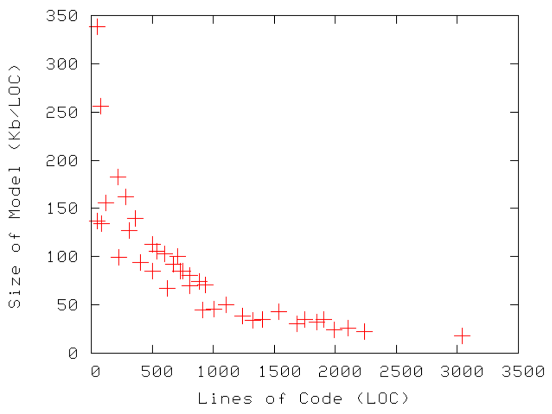


Figure 10: Lines of Code vs. Server Memory Usage.

For large software projects where there can be potentially millions of lines of code within a single revision, an alternative to an in-memory model might be required. In commercial settings, it is likely that specialised hardware can support multiple gigabytes of memory. In other situations where mass memory capabilities are not available, the CAISE architecture can easily be extended to incorporate an object-oriented database for models of potentially any size.

While the memory requirements for a CAISE-based project may seem significant, it is important to note that no other demands are placed on memory resources throughout the entire development environment. Unlike other architectures including IDEs, each CSE tool can rely on the CAISE server for all parsing, analysing and modelling of the software; tools themselves do not need to store a replica model.

6.2 Network Load

The design of the architecture ensures that network loads are as low as possible, and recent analysis of traffic verifies that for small user groups, no considerable strain is placed on a 100 Mbps Ethernet local area network. Even as the number of concurrent users increases to that of large development teams, today's networks are capable of accommodating the load.

When testing on wide area networks, the data throughput requirements are low enough for clients to be connected to the server from dial-up networks, but the latency can cause edit delays of up to several seconds. To support low speed wide area network connections, an alternative distributed system might be necessary where the anticipated results of modification requests are immediately displayed in the originating tool's display. In this case, a synchronisation routine will be required to run in a separate thread to resolve any modification discrepancies between tools.

At present, fault tolerance within CAISE has not been addressed. User trials and experimentation has been limited to within local networks, where error rates are low and are readily addressed by underlying communication protocols. For high-latency, high error network configurations such as intercontinental real-time software development, techniques for fault tolerance may need to be identified.

6.3 Response Times, Users and Model Size

From performance analysis, we are confident that as the number of users and the size of the project model increases, response times will remain stable. The direct impact of increased numbers of concurrent users within a CAISE project has been observed to be negligible; the number of connected users or opened files does not have a noticeable effect on server memory usage or response time. If all users are highly active at the same time the server response times will slow down, but in reality this is a very unlikely scenario.

Even if a given project has a very large semantic model, this does not necessarily affect the response times of the server. Most operations such as adding a new method to a class or querying the model for a specific relationship only require the traversal of a fixed subset of the entire model space. Therefore, even as the model grows in size, the response times should stay approximately constant.

7 Summary

Real-time support for CSE is an important emerging field of research. The size and complexity of today's software projects far exceeds the ability of conventional single-user tools to provide an environment of fine-grained communication between developers. While source code repository systems provide a degree of control over constantly evolving software projects, there is both the demand and enabling technology for more comprehensive tool support.

To date, the large development cost in constructing new CSE tools has been a major obstacle within the field of research. In this paper, we have shown how new CSE tools can be developed rapidly within the CAISE architecture.

We have presented example CAISE-based tools, discussed the underlying protocol that ensures tool synchronisation, and have illustrated how existing multi-user widgets can be utilised within existing software engineering tools. We have also discussed the CAISE Tool API at a level of detail suitable to provide insight for potential tool developers. Finally, we have addressed various performance issues and have reasoned why CAISE-based CSE tools are able to operate satisfactorily under a range of workloads.

The CAISE architecture and associated tools performed well during recent empirical and anecdotal evaluations. After illustrating the construction and use of CAISE-based CSE tools in this paper, it is hoped that others are encouraged to develop similar tools to support the emerging field of real-time CSE.

References

Carasik, R. P. & Grantham, C. E. (1988), A Case Study of CSCW in a Dispersed Organization, *in* 'CHI '88: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems', ACM Press, New York, NY, USA, pp. 61–66.

Churcher, N. & Cerecke, C. (1996), GroupCRC: Exploring CSCW Support for Software Engineering, *in* 'Proceedings of the 4th Australasian Conference on Computer-Human Interaction', IEEE Computer Society Press, Hamilton, New Zealand.

Cook, C. (2005), Towards Computer-Supported Collaborative Software Engineering, PhD thesis, University of Canterbury, Christchurch, New Zealand. Work in Progress.

Cook, C. & Churcher, N. (2003), An Extensible Framework for Collaborative Software Engineering, *in* D. Azada, ed., 'Proceedings of the Tenth Asia-Pacific Software Engineering Conference', IEEE Computer Society, Chiang Mai, Thailand, pp. 290–299.

Cook, C. & Churcher, N. (2005*a*), A User Evaluation of Synchronous Collaborative Software Engineering Tools, Technical Report TR-COSC 04/05, Department of Computer Science, University of Canterbury, Christchurch, New Zealand.

Cook, C. & Churcher, N. (2005*b*), Modelling and Measuring Collaborative Software Engineering, *in* V. Estivill-Castro, ed., 'Proceedings of ACSC2005: Twenty-Eighth Australasian Computer Science Conference', Vol. 38 of *Conferences in Research and Practice in Information Technology*, ACS, Newcastle, Australia, pp. 267–277.

Cook, C., Churcher, N. & Irwin, W. (2004), Towards Synchronous Collaborative Software Engineering, *in* 'Proceedings of the Eleventh Asia-Pacific Software Engineering Conference', IEEE Computer Society, Busan, Korea, pp. 230–239.

Cox, D. & Greenberg, S. (2000), Supporting Collaborative Interpretation in Distributed Groupware, *in* 'Proceedings of the ACM Conference on Computer Supported Cooperative Work', ACM Press, Philadelphia, PA, pp. 289–298.

Froehlich, J. & Dourish, P. (2004), Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams, *in* '6th International Conference on Software Engineering (ICSE'04)', IEEE, Edinburgh, Scotland, United Kingdom, pp. 387–396.

Graham, N., Stewart, H., Ryman, A., Kopaee, R. & Rasouli, R. (1999), A World-Wide-Web Architecture for Collaborative Software Design, *in* 'Software Technology and Engineering Practice', IEEE, Pittsburgh, Pennsylvania, pp. 22–32.

Greenberg, S. (1989), The 1988 Conference on Computer-Supported Cooperative Work: Trip Report, *in* 'SIGCHI Bulletin', Vol. 20 of *5*, ACM, pp. 49–55. Also published in Canadian Artificial Intelligence, **19**, April 1989.

Hill, J. & Gutwin, C. (2003), Awareness Support in a Groupware Widget Toolkit, *in* 'Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work', ACM Press, Sanibel Island, Florida, USA, pp. 256–267.

Lewis, S. (2005), 'Eclipse Communication Framework', Internet Homepage.
`http://www.eclipse.org/ecf/goals.html`

Reeves, M. & Zhu, J. (2004), Moomba A Collaborative Environment for Supporting Distributed Extreme Programming in Global Software Development, *in* J. Eckstein & H. Baumeister, eds, 'Lecture Notes in Computer Science', Vol. 3092, Springer-Verlag, pp. 38–50.

Roseman, M. & Greenberg, S. (1996), 'Building Real Time Groupware with GroupKit, A Groupware Toolkit', *ACM Transactions on Computer-Human Interaction* **3**(1), 66–106.

Sarma, A. & van der Hoek, A. (2002), Palantír: Coordinating Distributed Workspaces, *in* '26th Annual International Computer Software and Applications Conference', IEEE, Oxford, England.

Schummer, T. (2001), Lost and Found in Software Space, *in* '34th Annual Hawaii International Conference on System Sciences', IEEE Computer Society, Maui, Hawaii.

Plagiarism Detection across Programming Languages

Christian Arwin **S.M.M. Tahaghoghi**

School of Computer Science and Information Technology
RMIT University, GPO Box 2476V, Melbourne 3001, Australia.
{carwin,saied}@cs.rmit.edu.au

Abstract

Plagiarism is a widespread problem in assessment tasks; in computing courses, students often plagiarise source code. For all but the smallest classes, manual detection of such plagiarism is impractical, and, while automated tools are available, none has been applied to detect *inter-lingual plagiarism*, where source code is copied from one language to another. In this work, we propose a novel approach, XPlag, to detect plagiarism involving multiple languages using intermediate program code produced by a compiler suite. We describe experiments to evaluate XPlag, and show that we can detect inter-lingual plagiarism with reasonably good precision.

Keywords: program source code similarity, plagiarism detection

1 Introduction

Plagiarism — the representation of another's work as one's own — is a serious problem for academics; a survey performed by Sheard, Dick, Markham, Macdonald & Walsh (2002) on a sample of students at Monash and Swinburne universities shows that 85.4% of 137 Monash University students and 69.3% of 150 Swinburne University students admitted to having engaged in academic dishonesty. Educational institutions commonly attempt to reduce the incidence of plagiarism by applying penalties for violation of rules on academic dishonesty, yet plagiarism remains widespread (Zobel & Hamilton 2002).

Many computing courses have assessment tasks that require submission of program source code; students may plagiarise by copying code from friends, the Web, or so-called "private tutors" (Zobel 2004). For large cohorts, manual comparison of submissions to identify plagiarism is impractical, and so students may feel confident that their work will escape detection. There are also commercial concerns; organisations may be unknowingly liable to litigation for unauthorised use of program source code. Robust coderivative detection methods are essential.

Several approaches have been proposed for detecting code plagiarism; most use source code tokenisation and string matching algorithms to measure similarity. These generally perform well in detecting plagiarism that involves common disguising techniques such as statement reordering or modification of constant values and variable names (Gitchell & Tran 1999, Prechelt, Malpohl & Philippsen 2000, Wise 1996).

However, these approaches are designed and applied to detect plagiarism between programs written in a single language, and cannot handle cases where source code is copied from one language to another. We propose the names *intra-lingual plagiarism* for the former, and *inter-lingual plagiarism* or *cross-lingual plagiarism* for the latter.

We hypothesise that plagiarised programs, regardless of the language they are written in, have a similar code structure. We propose two solutions to address inter-lingual plagiarism. One is to compare the tokens produced by most existing plagiarism detection approaches that support more than one language. The second is to compare the intermediate code produced by a *compiler suite*, that is, a compiler that supports more than one language. The former typically supports only a few languages and requires a scanner and parser for each language, while the latter solution relies on the components of an existing generic compiler suite. We focus on the second approach in this work.

We test our new approach against three collections using ground truth developed from an existing state-of-the-art plagiarism detection system and through manual comparisons. The results show that our approach detects plagiarism with reasonably good precision for all the test collections. More importantly, it succeeds in detecting plagiarism across languages.

The remainder of this thesis is organised as follows. In Section 2, we discuss the main computer-based approaches to plagiarism detection. In Section 3, we describe our approach for the detection of inter-lingual plagiarism. In Section 4, we describe our experiments and our analysis of the results. We conclude in Section 5 with a summary of our work and thoughts for future research.

2 Background

Two factors that complicate plagiarism detection are the abundance of available resources and the variety of techniques used to disguise the copied materials. A number of approaches have been proposed to detect plagiarism in text and in program source code; we briefly review some of these in this section.

2.1 Text Plagiarism Detection

Text plagiarism involves copying parts of manuscripts, papers, and documents. Hoad & Zobel (2003) explore the *ranking* and *fingerprinting* approaches for detecting plagiarism of text. These approaches have a common preprocessing stage that includes *case folding*, *stemming* (removing prefix/suffix from words), *stopping* (removing common words), and *term parsing* (removing whitespace, punctuation, and control characters from the document).

The ranking approach consists of two stages to find documents similar to a query. In the first stage, documents are indexed. In the second stage, terms in the query document are matched against the indexed terms of each collection document, and a similarity score is calculated. Documents are ranked by decreasing similarity score for presentation to the user. This approach relies on the use of an effective similarity function to determine the similarity score for each document (Hoad & Zobel 2003). The fingerprinting approach also uses the two stages used by the ranking approach. However, it compares document fingerprints rather than document terms.

2.2 Source Code Plagiarism Detection

The nature of program source code makes it difficult to apply simple text-based detection techniques. Copied code is typically altered to avoid detection. Whale (1986) lists thirteen techniques that students may use to disguise the origin of copied code; these are "changing comments, changing formatting, changing identifiers, changing the order of operands in expressions, changing data types, replacing expressions by equivalents, adding redundant statements, changing the order of time-independent statements, changing the structure of iteration statements, changing the structure of selection statements, replacing procedure calls by the procedure body, introducing non-structured statements, combining original and copied program fragments". We consider there to be one additional item: the translation of source code from one language to another, or *inter-lingual plagiarism*. For example, source code written in C may be copied across to an implementation in Java.

There are several existing approaches to detect code plagiarism. Prechelt et al. (2000) identify two main categories of automated plagiarism detection for program source code; these are *feature comparison* and *structure comparison*. We explain these below.

2.2.1 Feature Comparison

In feature comparison, the similarity of two programs is estimated from the similarity of various software metrics, such as the average number of characters per line, the number of comment lines, the number of indented lines, the number of blank lines, and the number of tokens (for example, keywords, operator symbols, and standard library module names). Jones (2001) proposes a feature comparison category that compares two programs based on three *profiles*:

Physical profile characterises a program based on its physical attributes, such as the number of lines, words, and characters.

Halstead profile characterises a program based on its token types and frequencies. These includes the number of token occurrences (N), the number of unique tokens (n), and volume ($N \log_2 n$).

Composite profile a combination of the physical profile and the Halstead profile.

To detect plagiarism, the profiles of each program are calculated, and then normalised. The similarity of two programs is estimated by computing the Euclidean distance between their profiles (Jones 2001).

Prechelt et al. (2000) note that systems that use feature comparison may be very insensitive (can easily be misled), or very sensitive (producing many false positives) since they ignore program structure. Offenders may easily add or remove comments, variables, or redundant lines of code to escape detection (Chen, Li, McKinnon & Seker 2002, Prechelt et al. 2000, Whale 1990).

2.2.2 Structure Comparison

This approach relies on the belief that the similarity of two programs can be estimated from the similarity of their structure. Programs are compared in two stages; the first stage parses the code and generates token sequences, while the second stage compares the tokens. Three systems that fall into this category are `Sim` (Gitchell & Tran 1999), `YAP3` (Wise 1996), and `JPlag` (Prechelt et al. 2000).

Sim

`Sim` detects similarities between programs by evaluating their correctness, style, and uniqueness (Gitchell & Tran 1999). Each program is first parsed using the `flex` lexical analyser, producing a sequence of integers (tokens). The tokens for keywords, special symbols, and comments are predetermined, while the tokens for identifiers are assigned dynamically and stored in a shared symbol table; whitespace is discarded. The token stream of the second program is grouped into sections, each representing a module of the program; each section is separately aligned with the token stream of the first program. An alignment of two strings is performed by inserting spaces between characters to equalise their length. An alignment scoring scheme is used to calculate similarity. This rewards matches involving two identifiers by two points, and other matches by one point. It also penalises mismatches involving two identifiers by two points, and other mismatches by one point. Gaps also attract a two-point penalty. `Sim` can handle name changes and reordering of statements and functions.

YAP3

`YAP3` (Wise 1996) is another structure-based plagiarism detection system; it detects plagiarism through two phases. In the first phase, token sequences are generated from the source code. Comments and string constants are removed, and characters are converted to lower case. Functions are mapped to their base equivalents (such as `strncmp` to `strcmp`). In the second phase, the maximum, non-overlapping matches of the tokens sequences are then obtained using the running Karp-Rabin greedy string-tiling algorithm (Karp & Rabin 1987). `YAP3` is able to detect plagiarism with modified subsequences of lines and additional statements.

JPlag

`JPlag` (Prechelt et al. 2000) is a web-based detection system that uses a comparison algorithm similar to that of `YAP3`. In this system, the source code is parsed and converted into token strings. To minimise similarity by chance, JPlag includes some context of the program structure into the token strings, for example using the "`BEGIN_METHOD`" token to indicate an open brace at the beginning of a method and "`OPEN_BRACE`" to indicate other open braces. Whitespace, comments, and identifier names are ignored. The greedy string tiling algorithm is then used to compare token strings and identify the longest, non-overlapped common substrings. The result of the detection process is shown to the user in colour-coded HTML format.

Source Code

```
if (i==99) {
   int j=i+123;
}
```

Register Transfer Language (RTL)

```
...
(insn 14 12 15 (nil) (set (reg:CCZ 17 flags)
        (compare:CCZ (mem/f:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                    (const_int -4 [0xfffffffc])) [0 i+0 S4 A32])
            (const_int 99 [0x63]))) -1 (nil)
    (nil))

(jump_insn 15 14 16 (nil) (set (pc)
        (if_then_else (ne (reg:CCZ 17 flags)
                (const_int 0 [0x0]))
            (label_ref 22)
            (pc))) -1 (nil)
    (nil))

(note 16 15 17 0x4017d2c0 NOTE_INSN_BLOCK_BEG)

(note 17 16 19 NOTE_INSN_DELETED)

(insn 19 17 20 (nil) (parallel [
            (set (reg:SI 61)
                (plus:SI (mem/f:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                            (const_int -4 [0xfffffffc])) [0 i+0 S4 A32])
                    (const_int 123 [0x7b])))
            (clobber (reg:CC 17 flags))
        ]) -1 (nil)
    (nil))

(insn 20 19 21 (nil) (set (mem/f:SI (plus:SI (reg/f:SI 54 virtual-stack-vars)
                (const_int -8 [0xfffffff8])) [0 j+0 S4 A32])
        (reg:SI 61)) -1 (nil)
    (expr_list:REG_EQUAL (plus:SI (mem/f:SI
                    (plus:SI (reg/f:SI 54 virtual-stack-vars)
                    (const_int -4 [0xfffffffc])) [0 i+0 S4 A32])
            (const_int 123 [0x7b]))
        (nil)))
...
```

Optimised RTL

```
...
(insn 14 12 15 (nil) (set (reg:CCZ 17 flags)
        (compare:CCZ (reg/v:SI 61 [ i ])
            (const_int 99 [0x63]))) -1 (nil)
    (nil))

(jump_insn 15 14 16 (nil) (set (pc)
        (if_then_else (ne (reg:CCZ 17 flags)
                (const_int 0 [0x0]))
            (label_ref 21)
            (pc))) -1 (nil)
    (nil))

(note 16 15 17 0x4017d318 NOTE_INSN_BLOCK_BEG)

(note 17 16 19 NOTE_INSN_DELETED)

(insn 19 17 20 (nil) (parallel [
            (set (reg/v:SI 62 [ j ])
                (plus:SI (reg/v:SI 61 [ i ])
                    (const_int 123 [0x7b])))
            (clobber (reg:CC 17 flags))
        ]) -1 (nil)
    (nil))
...
```

Figure 1: An example of source code (top left), the corresponding unoptimised RTL (bottom left), and the optimised RTL (right).

3 Plagiarism Detection Using Intermediate Code

Existing plagiarism detection systems, whether based on feature or structure comparison, are developed to detect plagiarism in a particular language such as C, Java, Pascal, or Scheme. We refer to this as *intra-lingual plagiarism*. However, none has been applied to detect *inter-lingual plagiarism*, where code in one language is plagiarised and rendered in another.

In this section, we describe our novel approach, that we call XPlag, to detect inter-lingual plagiarism by comparing the structure of intermediate code produced by a compiler suite.

A compiler typically processes source code in two passes (Hernandez-Campos 2002). In the front end pass, a source code file is scanned (lexical analysis), parsed (syntax analysis), and semantically analysed to produce intermediate code. In the back end pass, the intermediate code is optimised and its binary code (executable) is generated.

Since we wish to detect plagiarism that involves multiple languages, we need a compiler that supports more than one language. We refer to this type of compiler as a *compiler suite*.

3.1 The Compiler Suite

Two popular compiler suites are Microsoft Visual Studio .NET and the GNU Compiler Collection (GCC). Microsoft Visual Studio .NET is based on the .NET framework[1] and supports many languages, among them Microsoft Visual C#, Visual Basic .NET, Visual J#, and Visual C++ .NET. Program source code is compiled first by the appropriate front-end compiler to produce the the common intermediate language (CIL) — also known as the Microsoft Intermediate Language (MSIL). At run time, a Just-In-Time (JIT) compiler is then used generate the executable (native) code from the intermediate code.

The popular GNU Compiler Collection (GCC)[2] also supports several languages including C, C++, Java, Fortran, and Objective C, and produces intermediate code in a common format (Jain, Sanyal & Khedker 2003). There is also ongoing work to integrate support for the .NET framework into GCC (Singer 2003). The GCC front end contains separate lexical analysis, syntax analysis, semantic analysis, and tree optimisation modules for each language; from this, a representation in a common intermediate code — the Register Transfer Language (RTL) — is generated. The back end optimises this RTL to produce machine code that is executable by the target machine.

The bulk of our work to date has focused on the GCC compiler suite using the C and Java languages, acknowledged to be the most popular[3] of those supported by this compiler suite.

3.2 The Register Transfer Language

The GCC Register Transfer Language[4] contains a series of instructions represented in nested parentheses. Each instruction contains a line number, a pointer to the previous instruction, and a pointer to the next instruction followed by expressions.

GCC provides three levels of compiler optimisation[5]. Figure 1 shows an example where the expressions in the optimised RTL — Figure 1 (right) — are simpler than the unoptimised RTL — Figure 1 (bottom left). In the optimised RTL, variable initialisation is performed by storing a value in a virtual register (represented by a `reg` token) rather than a virtual stack (represented by a `reg` and an offset value). This

[1]http://www.microsoft.com/net/basics/framework.asp

[2]http://gcc.gnu.org

[3]http://www.developer.com/lang/article.php/3390001

[4]http://gcc.gnu.org/onlinedocs/gcc-3.3.3/gccint/RTL.html

[5]http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

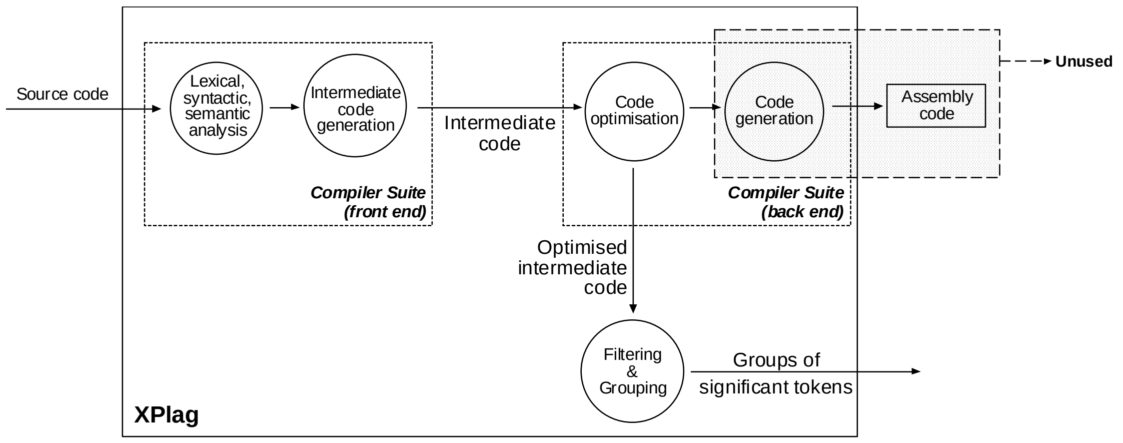

Source code
Lexical, syntactic, semantic analysis
Intermediate code generation
Compiler Suite (front end)
Intermediate code
Code optimisation
Code generation
Assembly code
Unused
Compiler Suite (back end)
Optimised intermediate code
Filtering & Grouping
Groups of significant tokens
XPlag

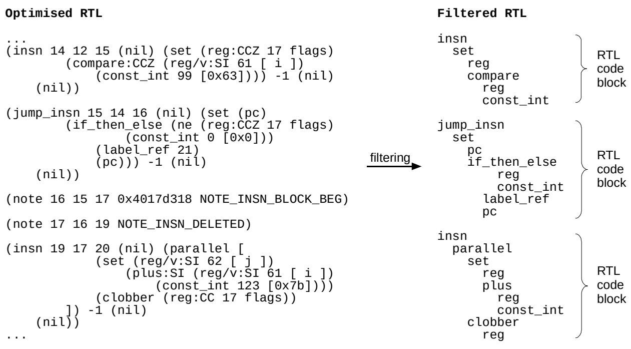
Figure 2: XPlag stages; the input source code is scanned and parsed by a compiler suite, producing the intermediate code. The intermediate code is optimised, filtered, and clustered, to produce a group of significant tokens.

insn	and	call	gt	truncate	NOTE_INSN_FUNCTION_BEG
const_int	parallel	call_insn	lt	ashiftrt	NOTE_INSN_FUNCTION_END
reg	clobber	call_placeholder	eq	ashift	NOTE_INSN_LOOP_BEG
set	plus	expr_list	le	lshiftrt	NOTE_INSN_LOOP_CONT
mem	minus	label_ref	ge	lshift	NOTE_INSN_LOOP_END_TOP_COND
code_label	unspec	symbol_ref	compare	sign_extend	NOTE_INSN_LOOP_END
use	jump_insn	pc	or	barrier	if_then_else

Table 1: The list of RTL keywords we consider significant.

makes the RTL file — and consequently the search engine index — more compact. Selecting optimisation also causes function inlining, where the compiler integrates functions shorter than a threshold (the default is 600 lines) into the calling code.

In our preliminary research, we determined that the highest optimisation level brings the most benefit to our approach, since the RTL instructions are simplified. Moreover, optimisation allows straightforward detection of cases where function inlining is used to disguise copied code (Whale 1986). GCC also provides a compilation option to add debugging information in the intermediate code. Our observations indicate that this additional information is not helpful, and so we do not report experiments using this option.


filtering
RTL code block
RTL code block
RTL code block

```
Optimised RTL

...
(insn 14 12 15 (nil) (set (reg:CCZ 17 flags)
        (compare:CCZ (reg/v:SI 61 [ i ])
            (const_int 99 [0x63]))) -1 (nil)
    (nil))

(jump_insn 15 14 16 (nil) (set (pc)
        (if_then_else (ne (reg:CCZ 17 flags)
                (const_int 0 [0x0]))
            (label_ref 21)
            (pc))) -1 (nil)
    (nil))

(note 16 15 17 0x4017d318 NOTE_INSN_BLOCK_BEG)

(note 17 16 19 NOTE_INSN_DELETED)

(insn 19 17 20 (nil) (parallel [
            (set (reg/v:SI 62 [ j ])
                (plus:SI (reg/v:SI 61 [ i ])
                    (const_int 123 [0x7b])))
            (clobber (reg:CC 17 flags))
        ]) -1 (nil)
    (nil))
...
```

```
Filtered RTL

insn
  set
    reg
    compare
      reg
      const_int
jump_insn
  set
    pc
    if_then_else
        reg
        const_int
      label_ref
      pc
insn
  parallel
    set
      reg
      plus
        reg
        const_int
    clobber
      reg
```

Figure 3: RTL example before filtering (left) and after filtering (right).

3.3 The XPlag approach

The XPlag mechanism comprises two stages. In the indexing stage, all programs in the collection are converted into tokens, and token information is stored in an inverted index. In the detection stage, source code is used to query the index and produce a list of programs in the collection, ranked by decreasing similarity.

The internal process of XPlag is illustrated in Figure 2. The source code input is scanned and parsed by a compiler suite, producing the intermediate code. After optimisation, the intermediate code is filtered and clustered, producing groups of overlapping tokens — n-grams — as the output. We follow with a detailed description of this approach.

3.4 The Filtering Process

The RTL of a program contains a sequence of instructions, each containing a set of keywords. Some keywords, such as variable names, register names, and constants, are insignificant for two reasons. First, they do not represent the structure of a program; second, variable names and constants can be altered to disguise plagiarism. We therefore consider these keywords to be stop words, and filter them from the optimised RTL[6]. The keywords we retain are listed in Table 1.

Figure 3 shows an example of RTL, before and after the filtering process. The indentation of the filtered RTL represents the depth of the nested expression in an RTL instruction. Constants, variable names, and machine modes — for example `SI`, indicating that the number is a single integer — are not retained. We discovered that filtered RTL for variable declarations, branching statements, and function calls have similar sequences of keywords across C and Java. However, we observed that the RTL structure of looping statements is different although the programs are similar. To address this, we keep the `NOTE_INSN_LOOP_BEG` and `NOTE_INSN_LOOP_END` keywords that are used to indicate the beginning and the end of a loop in the RTL generated from both languages.

3.5 The Mapping and Grouping Process

The significant keywords retained by the filtering process are between two and twenty-two characters, as listed in Table 1.

[6]We use the *flex* lexical analyser for this purpose.

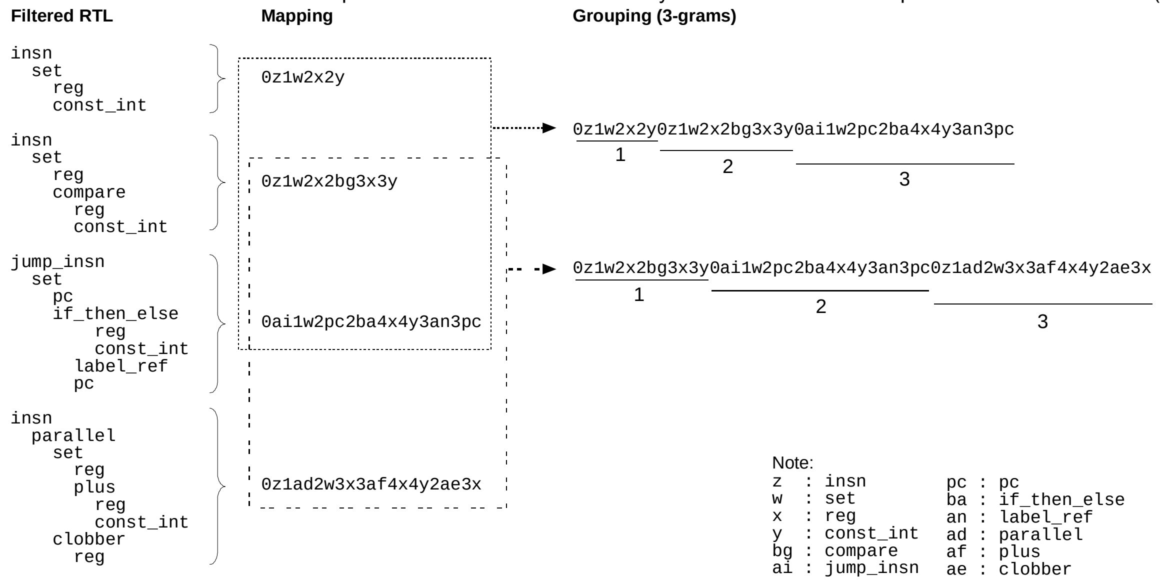


Figure 4: An example of the RTL mapping and grouping processes. Each significant keyword in the filtered RTL is converted to a one- or two-character code. Then, the mapped RTL is grouped into 3-grams. The number before each mapped RTL keyword represents the depth of indentation in the filtered RTL.

Two programs may contain similar instructions even where no plagiarism has occurred. Hence, the similarity of two programs should not be estimated by simply comparing the pairs of instructions contained in both programs. XPlag groups instructions into n-grams, where each gram contains n instructions, and each instruction contains a set of significant keywords.

We consider each RTL code block as a gram, with the indentation level indicated by a number. Figure 4 illustrates how the optimised RTL representation of a program is mapped and then grouped into 3-grams. The value of n is chosen empirically — we discuss this in further detail in Section 4.

3.6 The Search Engine

Plagiarism detection systems that are based on pair-wise comparisons are not scalable; a good alternative approach is to index and query the tokens with a search engine (Burrows, Tahaghoghi & Zobel 2004). We incorporate a search engine into XPlag to perform two tasks:

1. In the indexing stage, the grouped RTL of the source code files in the collection is indexed.
2. In the detection stage, the grouped RTL of the query source code is used to search the index, and the programs of the collection are listed ranked by decreasing similarity to the query.

XPlag consists of two stages, namely the *indexing* and *detection* stages. In the indexing stage, a collection of programs is compiled with the compiler suite using the highest optimisation option, producing the intermediate code. This is then filtered, mapped, and grouped to n-grams, producing groups of significant keywords that are then indexed by the search engine. In the detection stage, the source code to be checked is similarly compiled, filtered, mapped, and grouped to n-grams, producing a group of significant intermediate code keywords. These keywords are then used as a query to the search engine, returning a list of similar programs ordered by decreasing similarity.

The similarity between a collection program and the query is estimated using a similarity measure or ranking function. The Okapi BM25 similarity function is highly effective for general text search, and is defined as (Robertson & Walker 1999):

$$\sum_{t \in Q} w_t \cdot \frac{(k_1+1)\, f_{d,t}}{K + f_{d,t}} \cdot \frac{(k_3+1)\, f_{q,t}}{k_3 + f_{d,t}}$$

where:

$$w_t = \log_e \left(\frac{N - f_t + 0.5}{f_t + 0.5} \right), K = k_1 \cdot \left((1-b) + \frac{b \cdot W_d}{W_D} \right)$$

W_d	=	document length	K_1	=	1.2
W_D	=	average document length	k_3	=	infinite
N	=	documents in collection	b	=	0.75
$f_{q,t}$	=	query-term frequency	f_t	=	collection frequency
$f_{d,t}$	=	within-document frequency			

BM25 is more suited to retrieval of text documents rather than to code, since the more often a query term occurs in a document, the higher the score given to the document. Chawla (2003) proposes the PlagiRank ranking function as more appropriate for code retrieval. This gives a higher score to documents in which query terms have the same frequency:

$$Score(Q, Q_d) = \sum_{t \in Q \cap D d} \left(\ln \frac{f_{qt}}{f_{dt}} + 1 \right) \cdot f_{qt} \cdot \frac{1}{W_d W_q} \text{where} f_{qt} \leq f_{dt}$$

$$Score(Q, Q_d) = \sum_{t \in Q \cap D d} \left(\ln \frac{f_{dt}}{f_{qt}} + 1 \right) \cdot f_{qt} \cdot \frac{1}{W_d W_q} \text{where} f_{qt} \geq f_{dt}$$

We evaluate our approach using both these measures and discuss the results in Section 4.

3.6.1 The Indexing and Detection Stages

In the indexing stage, the n-grams of the RTL of each program in the collection are collated into the TREC format[7], and then indexed by the search engine.

In the detection stage, the RTL n-grams of the program to be checked are run as a query by the search engine, and a list of programs similar to the queried program is returned. We examine the top twenty documents.

The search engine provides an absolute similarity score for each document. This is unlikely to be meaningful to the average user, and so we instead refer to the Relative Percentage Similarity (RPS); this is the

[7]http://www.seg.rmit.edu.au/zettair/zettair/doc/Readme.html

ratio between the similarity score of a source code document and the similarity score of the query document to itself. To calculate this, the query source code must also be indexed by the search engine, and will be returned as the first answer. This represents the perfect match, with an RPS of 100%.

$$\text{Relative Percentage Similarity } (RPS)_i = \begin{cases} 100\% & \text{if } i = 1 \\ S_i/S_1 & \text{if } i > 1 \end{cases}$$

where i, RPS_i, and S_i represent the rank, Relative Percentage Similarity of the i-th document, and the search engine score of the i-th document, respectively. For example, if the search engine score of program file `0.c` is 0.5 and the score of `67.c` is 0.4328, then the relative percentage similarity of `0.c` is 100% (because $i = 1$), and the score of program file `67.c` is 0.4328/0.5=86.55%.

4 Experiments and Analysis

We continue with a discussion of our experiments, our analysis, and the external baseline we use to evaluate our technique.

All source code in our experiments was compiled using GCC version 3.3.3 on an Intel Pentium IV 2.4 GHz processor running the Linux SuSe 9.1 operating system. We also used version 0.6.1 of the Zettair[8] search engine developed by the RMIT University Search Engine Group.

The aim of our experiments is twofold. First, to evaluate the performance of XPlag in detecting plagiarism among programs written in one particular language, that is, intra-lingual plagiarism. Second, to evaluate the performance of XPlag to detect interlingual plagiarism. For this purpose, we use three different collections of program source code:

1. `Collection-C`: contains 79 C programs from student submissions for an assignment in a course on Secure Electronic Commerce offered in Semester 2 2003.
2. `Collection-J`: contains 107 Java programs from the same assignment as `Collection-C`.
3. `Collection-X`: contains 206 programs from the combination of `Collection-C` and `Collection-J` and with the addition of ten equivalent pairs of C and Java program files from the Web. This collection was used to see how well XPlag can detect plagiarism across C and Java programs.

Some programs in `Collection-X` were obtained by translating programs written in the C language to the Java language using the `Jazillian` online translation tool[9] to reflect an approach students may take when copying. We could not find any translation tools to translate programs written in the Java language to the C language. We found that some minor editing is still required to allow the code translated by `Jazillian` to be compiled under GCC. For example, we have to add the '`static`' keyword before each function called from the static `main()` function, and replace '`int int`' to '`int`' because `Jazillian` translates '`unsigned int`' as '`int int`'.

There are other automatic C-to-Java translation programs available[10], but most, such as `C2J` and `Ephedra` produce output that is clearly machine-generated code.

[8] `http://www.seg.rmit.edu.au/zettair/`

[9] `http://www.jazillian.com`

[10] `http://www.jazillian.com/competition.html`

4.1 Ground Truth and Evaluation

To evaluate the effectiveness of our approach, we need the ground truth, that is, a list of programs that are known to be plagiarised. The ground truth of each set was determined as follows:

1. For `Collection-C`, we used exhaustive manual comparisons. There were twelve groups of programs that we regarded as copied.
2. For `Collection-J`, because of time constraints, we manually verified pairs of similar programs identified by JPlag. We found seven groups of plagiarised programs.
3. For `Collection-X`, we used the known cross-plagiarised programs downloaded from the Web, and the pairs which we translated using `Jazillian`.

The difference in the way the ground truth for each collection was prepared is likely to affect the absolute performance of JPlag and XPlag. Nevertheless, we believe that the experimental results reflect the relative performance of the two approaches within each collection.

To quantify the performance of our detection, we use the standard information retrieval measures of *precision* and *recall* (Witten, Moffat & Bell 1999). *Precision* is the ratio of documents retrieved that are relevant, while *recall* is the proportion of the relevant documents that have been retrieved.

$$\text{Precision (P)} = \frac{\text{relevant documents retrieved}}{\text{retrieved documents}}$$

$$\text{Recall (R)} = \frac{\text{relevant documents retrieved}}{\text{relevant documents}}$$

In the context of source code plagiarism detection, precision represents the number of plagiarised programs at some point in the returned list. The higher the precision, the more accurate the detection (fewer false positives). Recall represents the number of plagiarised programs detected out of all plagiarised programs in the collection. The higher the recall, the fewer copied programs escape detection (fewer false negatives).

Three measures derived from precision and recall include *R-precision*, *Precision@n*, and *interpolated precision-recall*. *R-precision* is the precision at the R-th program on the list, where R is the number of correct answers for the query; *Precision@n* is the precision at the n-th program on the list; finally, the interpolated precision at a particular recall level is the highest precision observed at that or any higher recall level. Interpolated precision-recall is usually shown at the eleven 10% steps of recall from 0% to 100%.

Important to us are the Precision@2 (P@2), Precision@5 (P@5), Precision@10 (P@10), R-Precision (R-P), and interpolated precision-recall scores. Precision@2 is useful for measuring how effective XPlag ranks a copied program at the second position on the list. Since the first program on the list is always the query program, Precision@2 is effectively the precision when only the most similar collection document to the query is examined. Precision@5 is used to evaluate the accuracy of XPlag in returning the top five programs; this is a useful cut-off point, assuming that a program is unlikely to have more than four other coderivatives in the collection.

We also evaluate the precision and recall values at every 5% of the Relative Percentage Similarity (RPS) to estimate an RPS value that can be used by users as a good cut-off value to stop manual verification,

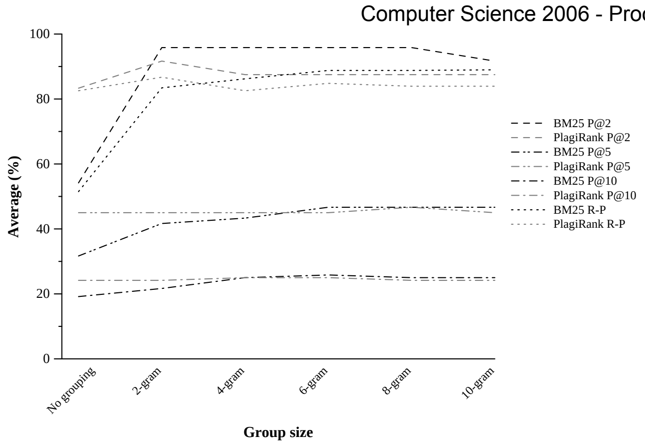


Figure 5: Performance comparison of the BM25 and PlagiRank similarity measures for varying n-gram sizes on Collection-C.

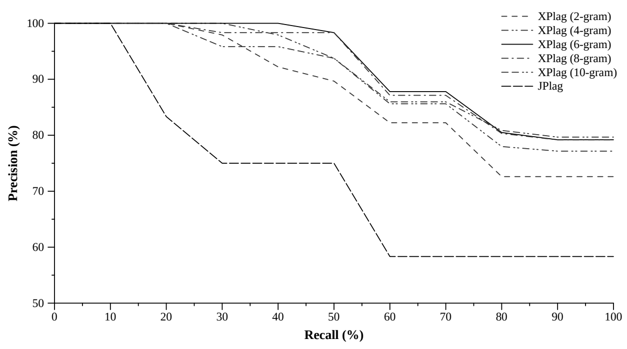


Figure 6: Interpolated precision-recall using the BM25 similarity measure and various grouping sizes on Collection C.

although this is likely to be somewhat dependent on the preference of the user for high precision or for high recall.

To test whether differences in performance are significant, we use the Wilcoxon signed rank test at the 95% confidence level.

4.2 Experiments with Collection-C

In our first series of experiments, we aim to evaluate the performance of XPlag in detecting plagiarism involving only C programs (`Collection-C`), identify the best grouping size to be used, and investigate which ranking function, BM25 or PlagiRank, produces better results. Our query set contains twelve programs taken from our ground truth. We find that grouping keywords into n-grams improves the precision of XPlag, as shown in Figure 5. Although we initially expected the PlagiRank similarity measure, specifically designed for plagiarism detection, to outperform BM25, the reverse is true for most combinations of n-gram sizes and evaluation measurements. Interestingly, all XPlag results are better than the JPlag baseline.

In our experiment, the BM25 with a group size of 6 produces the highest precision of all evaluation measurements. Figure 6 shows the interpolated precision at standard recall levels using the BM25 and 6-grams. We see that precision drops sharply after 50% recall (when half the incidents of known plagiarism instances have been retrieved), but remains above 75% for most n-gram sizes tested.

For comparison, we performed plagiarism detection using JPlag with the sensitivity value — or the minimum match length — set to 6; this value is equivalent to the grouping size of 6 that we used for XPlag.

Average Precision at	XPlag (%)	JPlag (%)
2	100.00	79.00
5	44.00	32.00
10	22.00	16.00
R	80.00	71.00

Table 2: Performance comparison of XPlag and JPlag on Collection-C.

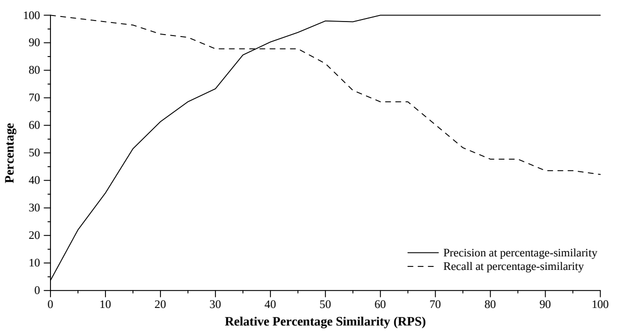


Figure 7: Average precision and recall values at 5% intervals of Relative Percentage Similarity using the BM25 and 6-grams on Collection-C.

Table 2 shows that XPlag outperforms JPlag in all evaluation measurements. Furthermore, the interpolated precision of XPlag at standard recall levels is significantly higher than JPlag, as shown in Figure 6. Table 2 also shows that XPlag accurately returns copied C programs at the second position on the list (Precision@2 is 100%).

In application, manual examination of highly ranked documents is likely to be impractical, and so we explore how the Relative Percentage Similarity value relates to the tradeoff between recall and precision. We plot the average precision and recall values at 5% Relative Percentage Similarity intervals in Figure 7. At 0% RPS, the recall reaches 100% (because all programs in the collection are listed) and the average precision is around 4% (because many false positives are returned). When RPS is equal to or greater than 60%, precision reaches 100% (no false positives), but recall decreases from 66% (at 60% RPS) to 48% (at 100% RPS). Requiring matches to have a higher RPS results in more false negatives.

4.3 Experiments with Collection-J

In our next series of experiments, we evaluated the performance of XPlag in detecting plagiarism involving only Java programs (`Collection-J`). We inspected the JPlag detection result to obtain groups of copied programs, and verified seven program pairs as copied.

Using the copied pairs as the query set, we find the result to be consistent with our experiments on `Collection-C`. Figure 8 demonstrates that grouping improves the performance of XPlag, and that BM25 produces better results than PlagiRank for most combinations of grouping sizes and evaluation measurements. The figure also shows that BM25 with group sizes of 4 and 6 produces the highest precision for all evaluation measures, although Figure 9 shows that using 4-grams leads to the highest interpolated precision results.

Table 3 compares the performance of XPlag and JPlag for detection of plagiarism among Java programs. Since the ground truth of `Collection-J` was generated from the JPlag detection result, JPlag performance represents ideal performance here; this is shown as a constant 100% interpolated precision-recall in Figure 9. While XPlag appears to perform

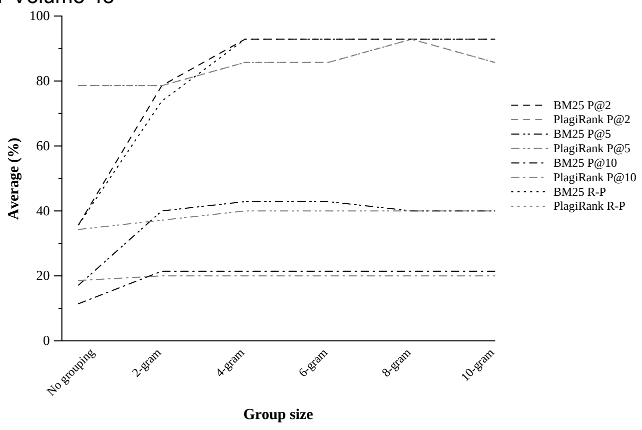


Figure 8: Performance comparison of the BM25 and PlagiRank similarity measures for varying n-gram sizes on Collection-J.

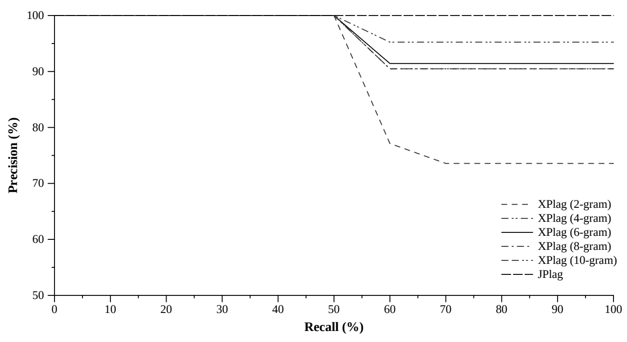


Figure 9: Interpolated precision-recall using the BM25 similarity measure and various grouping sizes on Collection J.

less than this ideal, the difference is statistically insignificant[11]. XPlag successfully returns all occurrences of plagiarism in the collection within the first five answers — the Precision@5 is equal for both XPlag and JPlag — and returns copied Java programs at the second position on the list with an average precision of 92.86%.

Figure 10 shows that the precision increases at a constant rate from 10% RPS, and reaches 100% at 55% RPS. From 25% RPS onwards, recall decreases gradually from 100% to 48% at 100% RPS. We observe that for both `Collection-C` and `Collection-J`, precision is greater than 90% (less than 10% false positives) and recall is greater than 80% (less than 20% false negatives) at 50% RPS.

4.4 Experiments with Collection-X

We have shown that XPlag can detect intra-lingual plagiarism with reasonable precision and recall values. To investigate the XPlag performance in detecting inter-lingual plagiarism, we use `Collection-X` and two different query sets: `Queryset-C` (containing only the ten C programs); and `Queryset-Java` (containing only the ten Java programs).

To explore whether we can use JPlag again as our baseline, we tried to perform plagiarism detection on `Collection-X` using the JPlag C/C++ and Java parsers in turn. However, neither was able to reveal inter-lingual plagiarism. The detection using the Java parser excludes 97 programs due to unsuccessful compilation; this is understandable since the Java parser cannot process the submitted C programs. In

Average Precision at	XPlag (%)	JPlag (%)
2	92.86	100.00
5	42.86	42.86
10	21.43	21.43
R	92.86	100.00

Table 3: Performance comparison of XPlag and JPlag on Collection-J.

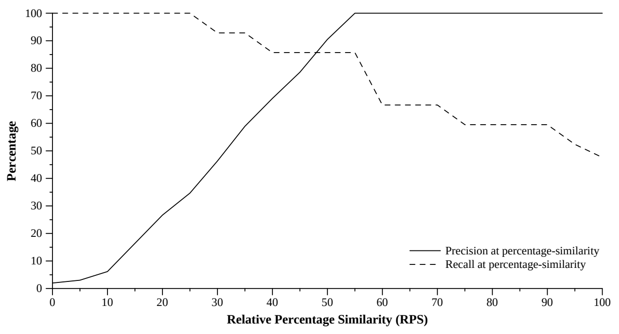


Figure 10: Average precision and recall values at 5% interval of Relative Percentage Similarity using the BM25 and 4-grams on Collection-J.

contrast, although the C parser can process most of the Java programs, only one incidence of inter-lingual plagiarism is reported, and that because one of the subroutines in both programs is identical. While this is not surprising — after all, JPlag is not designed to detect inter-lingual plagiarism — it leaves us with no external baseline for `Collection-X`.

Figure 11 and Figure 12 show that BM25 with a group size of 2 produces the best Precision@2 and interpolated precision-recall for both query sets. Using `Queryset-C`, precision increases sharply from 0% to 30% RPS, as shown in Figure 13 (a); while using `Queryset-Java`, precision increases sharply from 0% to 40% RPS, as shown in Figure 13 (b). Precision reaches 100% for both query sets when RPS is equal to or greater than 60%, while recall is above 90% when RPS is equal to or less than 20%.

5 Discussion and Future Work

Plagiarism is a serious and widespread problem. Several approaches have been proposed to reveal plagiarism in source code, but these only aim to detect plagiarism involving one programming language. In this paper, we have described our novel approach, XPlag, to detect inter-lingual plagiarism by inspecting the intermediate code produced by a compiler suite. Using three different collections and employing the popular JPlag system as our baseline, we have shown that XPlag can detect intra-lingual plagiarism in and Java programs with reasonably good precision. Significantly, we have also shown that XPlag can detect inter-lingual plagiarism, albeit with lower accuracy than for intra-lingual plagiarism.

While the RTL for variable declarations, function calls, and branching statements of C and Java programs are similar, the RTL of Java programs often contains instructions for processing classes and function calls of the standard Java library. For example, there are fewer RTL instructions for the C '`printf()`' function call than the Java '`System.out.println()`' method call. We believe that the performance of XPlag can be improved by enhancing the filtering process to remove insignificant instruction groups, and by classifying equivalent function calls in C and Java RTL.

There are some limitations to our approach that

[11]We used the Wilcoxon signed rank test at the 95% confidence level to compare the interpolated precision at standard recall levels of XPlag (4-grams) and JPlag.

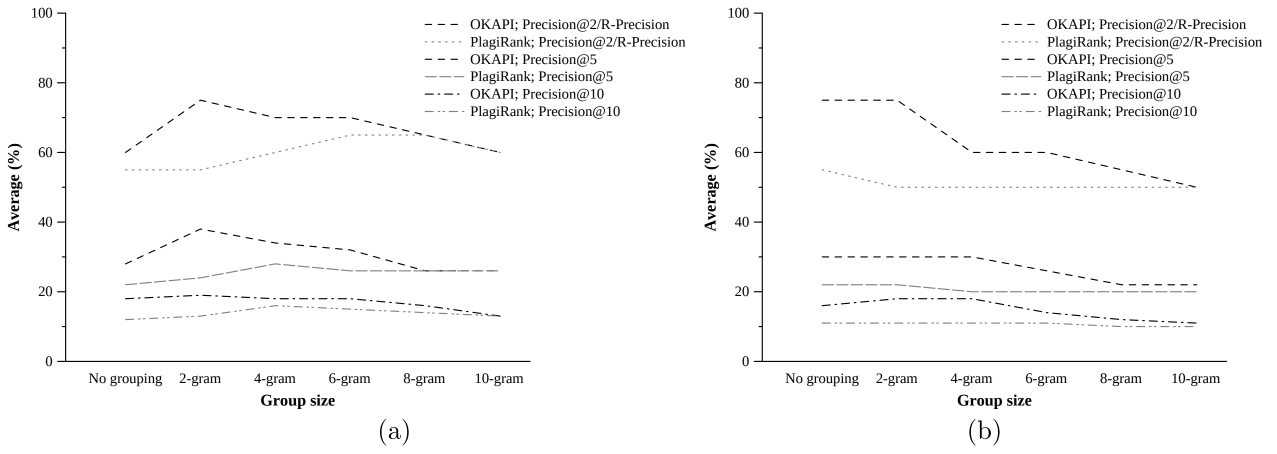


Figure 11: Performance comparison of the BM25 and PlagiRank similarity measures for varying n-gram sizes on Collection-X using (a) Queryset-C and (b) Queryset-Java.

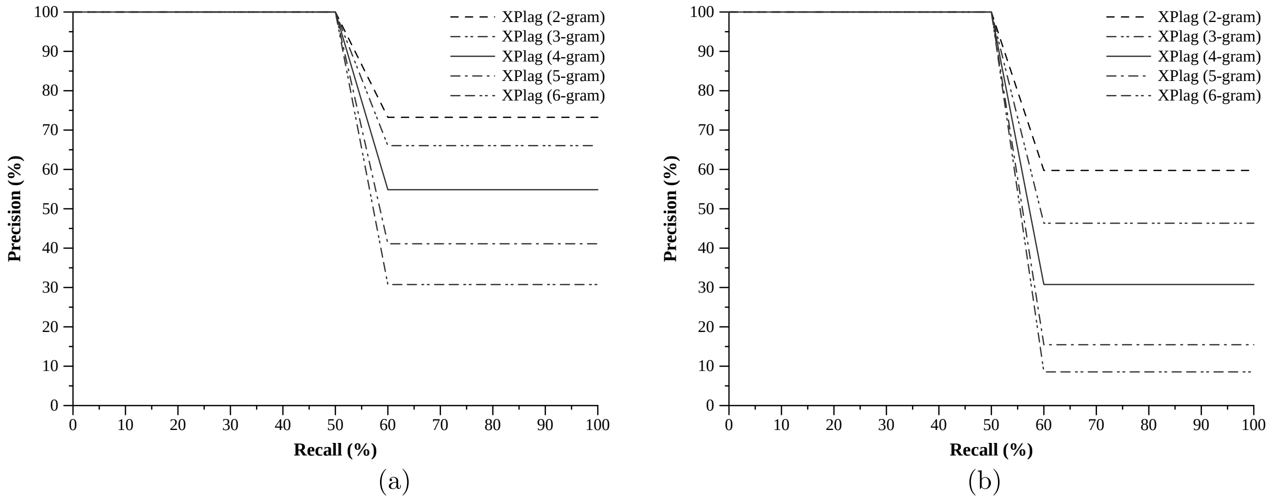


Figure 12: Interpolated precision-recall using the BM25 similarity measure and various grouping sizes on Collection X using (a) Queryset-C and (b) Queryset-Java.

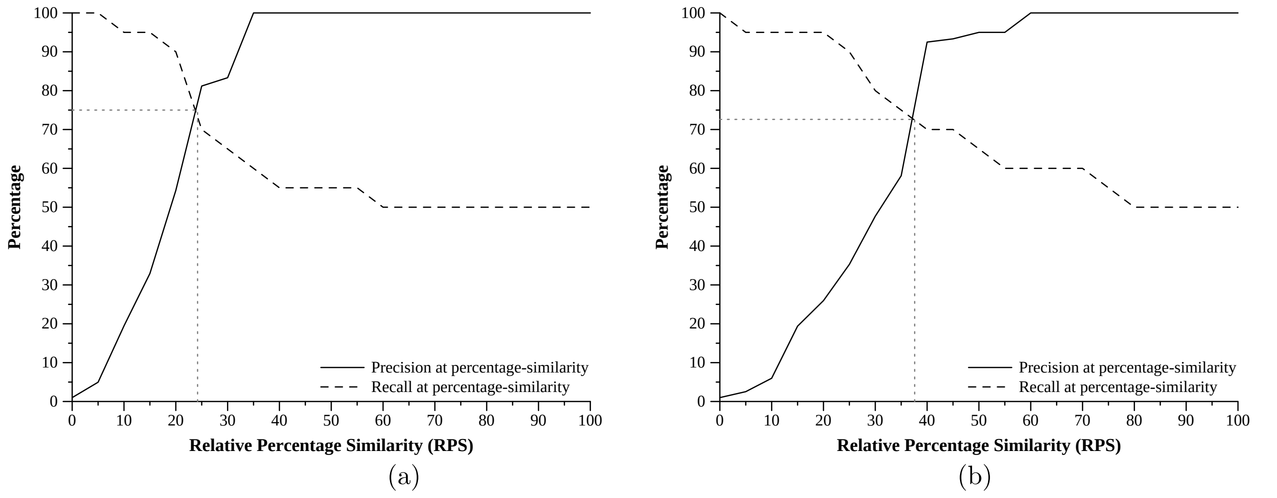


Figure 13: Average precision and recall values at 5% interval of Relative Percentage Similarity using the BM25 and 2-grams on Collection-X using (a) Queryset-C and (b) Queryset-Java.

should be addressed in future work. First, all submitted programs must be successfully compiled by the compiler suite; if a program cannot be successfully compiled, it must be corrected manually for further processing. In an educational setting, this might perhaps be addressed by penalising non-compilable code as a matter of assessment policy.

Second, the results of our experiments are only valid when using the GCC compiler suite for plagiarism detection involving programs written in the C and Java languages. Preliminary experiments indicate that the intermediate language produced by the Microsoft Visual Studio .NET compiler suite can be used for intra-lingual plagiarism detection. We plan detailed experiments with this compiler suite.

Third, while the collections we used are realistic for a typical computing courses, we must investigate how the effectiveness of our approach behaves across other collections, and also for very large repositories of historical or crawled program source code. Associated work on intra-lingual plagiarism detection indicates that the underlying approach scales well in both effectiveness and efficiency (Burrows et al. 2004, Chawla 2003).

Finally, we plan to explore the alternative approach of using tokens produced by existing approaches (for example `Sim` or `JPlag`) instead of using intermediate code.

Overall, we believe that our approach can greatly help address the problem of inter-lingual plagiarism, and in this way help reduce the incidence of code plagiarism in general.

Acknowledgments

We thank Professor Justin Zobel and Guido Mapohl for their advice on this project, and Steven Burrows for donating `Collection-C`.

References

Burrows, S., Tahaghoghi, S. M. M. & Zobel, J. (2004), Efficient and effective plagiarism detection for large code repositories, *in* 'G. Abraham and B.I.P. Rubinstein Editors, Proceedings of the Second Australian Undergraduate Students' Computing Conference (AUSCC04)', pp. 8–15.

Chawla, M. (2003), An indexing technique for efficiently detecting plagiarism in large volumes of source code, Honours thesis, RMIT University, Melbourne, Australia, October.

Chen, X., Li, M., McKinnon, B. & Seker, A. (2002), 'A theory of uncheatable program plagiarism detection and its practical implementation'. URL: `http://www.cs.ucsb.edu/~mli/sid.ps` [13 August 2005].

Gitchell, D. & Tran, N. (1999), Sim: a utility for detecting similarity in computer programs, *in* 'Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education', ACM Press, pp. 266–270.

Hernandez-Campos, F. (2002), 'Lecture 31: Building a runnable program'. URL: `http://www.cs.unc.edu/~stotts/COMP144/lectures/lect31.pdf` [13 August 2005].

Hoad, T. & Zobel, J. (2003), 'Methods for identifying versioned and plagiarised documents', *Journal of the American Society of Information Science and Technology* **54**(3), 203–215.

Jain, N., Sanyal, A. & Khedker, U. (2003), Retargeting GCC for cradle's DSE processor, Technical report, Department of Computer Science & Engineering, Indian Institute of Technology, Bombay, Bombay, India.

Jones, E. L. (2001), Metrics based plagiarism monitoring, *in* 'Proceedings of the Sixth Annual CCSC Northeastern Conference, Middlebury, Vermont', pp. 1–8.

Karp, R. M. & Rabin, M. O. (1987), 'Efficient randomized pattern-matching algorithms', *IBM Journal of Research and Development* **31(2)**, 249–260.

Prechelt, L., Malpohl, G. & Philippsen, M. (2000), JPlag: Finding plagiarisms among a set of programs, Technical Report 2000-1, Fakultat fur Informatik Universität Karlsruhe, D76128 Karlsruhe, Germany.

Robertson, S. E. & Walker, S. (1999), Okapi/Keenbow at TREC-8, *in* 'The Eighth Text Retrieval Conference (TREC-8)', pp. 151–162.

Sheard, J., Dick, M., Markham, S., Macdonald, I. & Walsh, M. (2002), Cheating and plagiarism: Perceptions and practices of first year IT students, *in* 'Proceedings of the Seventh Annual Conference on Innovation and Technology in Computer Science Education', pp. 183–187.

Singer, J. (2003), GCC .NET—a feasibility study, *in* 'Proceedings of the First International Workshop on C# and .NET Technologies', University of West Bohemia, Plzen, Czech Republic.

Whale, G. (1986), Detection of plagiarism in student programs, *in* 'Proceedings of the Ninth Australian Computer Science Conference, Canberra', pp. 231–241.

Whale, G. (1990), 'Identification of program similarity in large populations', *The Computer Journal* **33**, 2.

Wise, M. J. (1996), 'YAP3: Improved detection of similarities in computer program and other texts', *SIGCSE Bulletin* **28**(1), 130–134.

Witten, I. H., Moffat, A. & Bell, T. C. (1999), *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann Publishers, second edition.

Zobel, J. (2004), "Uni cheats racket": a case study in plagiarism investigation, *in* 'Proceedings of the Sixth Conference on Australian Computing Education', Australian Computer Society, Inc., pp. 357–365.

Zobel, J. & Hamilton, M. (2002), 'Managing student plagiarism in large academic departments', *Australian Universities Review* **45**(1), 23–30.

Programming with Heterogeneous Structures: Manipulating XML data Using bondi

F. Y. Huang
School of Computing
Queen's University
huang@cs.queensu.ca

C. B. Jay
Faculty of Information Technology
University of Technology, Sydney
cbj@it.uts.edu.au

D. B. Skillicorn
School of Computing
Queen's University
skill@cs.queensu.ca

Abstract

Manipulating semistructured data, such as XML, does not fit well within conventional programming languages. A typical manipulation requires finding *all* occurrences of a structure matching a *structured* search pattern, whose context may be *different* in different places, and both aspects cause difficulty. If a special-purpose query language is used to manipulate XML, an interface to a more general programming environment is required, and this interface typically creates runtime overhead for type conversion. However, adding XML manipulation to a general-purpose programming language has proven difficult because of problems associated with expressiveness and typing.

We show an alternative approach that handles many kinds of patterns within an existing strongly-typed general-purpose programming language called bondi. The key ideas are to express complex search patterns as structures of simple patterns, pass these complex patterns as parameters to generic data-processing functions and traverse heterogeneous data structures by a generalized form of pattern matching. These ideas are made possible by the language's support for pattern calculus, whose typing on structures and patterns enables path and pattern polymorphism. With this approach, adding a new kind of pattern is just a matter of programming, not language design.

Keywords: Pattern Calculus, functional programming, heterogeneous data structure, XML processing

1 Introduction

When processing semistructured data such as XML, a basic operation is to locate data items by their position in a structured context, usually described by a pattern or sequence of patterns. In some situations, these patterns can be as simple as matching a single type of element, for example, in the search for all `population` elements in a geographical dataset, `population` is a simple pattern to match for target data items. In other situations, search patterns are more complex, but complex patterns can usually be decomposed into simpler ones. For example, in the search for the complex pattern `population of individual cities in Canada` includes searches for a `country` element with a `countryName` descendant element having the value `Canada`, and some `city` descendant elements which in turn have `population` descendant elements (we do not use attributes in our examples, because attributes can be transformed into elements easily).

There are two ways to compose simpler patterns into more complex ones. The first is *vertical* composition, as in the search for the population of cities of Canada. Such complex patterns match XML elements from different levels of a hierarchy. We call them vertical patterns. Location paths, expressed in the popular XPath (Clark & DeRose 1999) language, fall in this category. The second is *horizontal* composition, as in the search for cities having child elements for name, population, either timezone or continent, and zero or more rivers, i.e., the pattern `(cityName, population, timezone|continent, river*)`. Such complex patterns match XML elements from the same level of a hierarchy. We call them horizontal patterns.

Vertical and horizontal patterns can be combined into even more complex search patterns. For example, the pattern: contact phone numbers of city halls of Canadian cities having child elements for name, population and zero or more rivers, is a combination of vertical and horizontal patterns.

Semistructured data processing poses a problem for general-purpose programming languages. For example, typical processing of XML data consists of a *search* for all occurrences of a search pattern, *extraction* of the occurrences and some part of their context, *changes* to these extracted data structures, and their *replacement* in the entire data structure. General-purpose programming languages have trouble typing such programs because the target pattern can be complex in different ways and can occur in different contexts; and they also have trouble expressing the implied universal quantifier in the search.

These difficulties led to the design of special-purpose XML query languages, emerging both from the database community and the structured text community. The problem with using a query language for manipulating XML is that it creates an interface between data extraction and data use. For example, in a typical web environment, the data itself is in a back-end system and the results of the data query/transformation must be passed to a front-end system for further processing. The existence of a boundary requires a common format, usually quite a low-level one such as a string, by which the back-end and front-end communicate. This requires extra programming effort, subject to security holes and runtime overhead. This has been called the *impedance mismatch problem* (Bancilhon & Maier 1988, Wadler 2004).

There is an obvious benefit to extending general-purpose programming languages so that they can handle XML manipulation in native mode. Doing so reduces or eliminates the impedance mismatch problem, since computations at the browser, front-end, and back-end can all be done in the same language environment. Because such languages are typed, security of programs can be verified statically, reducing runtime overhead for dynamic type checking and the chance of catastrophic failure or unintended leakage of information. Also, because of the expressive power of such languages, programs may be smaller and more modular, making them cheaper and easier to build and maintain.

Extending general-purpose programming languages to include XML manipulation directly has proven difficult, although a number of attempts have made some progress towards this goal. Such attempts usually end up with a new or extended language that can only handle specific kinds of hard-coded XML search patterns which cannot be passed as typed parameters, and cannot be further extended without changing the language.

In this paper we show that an existing general-purpose functional-programming language, bondi, in which structures and patterns are treated as of equal importance to data and functions, allows XML manipulation to be expressed in a natural and general way, and without any extensions to the language.

Rather than aggregate the features found in the existing wide variety of XML query and transformation languages, bondi treats structures and patterns as first-class objects. Hence, control flow can be determined by structures, not just datum values; structures and structure-matching patterns are well typed and can be passed as parameters. This respect for the data creates new power for programming with heterogeneous data structures.

Because bondi is a general-purpose language, XML applications can be seamlessly integrated into other applications, including web and web service applications.

In this paper we show that:

- Existing approaches to XML processing only handle limited kinds of search patterns, with weaknesses in either type-safety, parameterizability, extensibility to other kinds of patterns, or all three;
- The ability to handle structures and structure-matching patterns in the same way as other programming entities is the key to manipulating XML in an effective, but also properly typed, way;
- This increase in expressiveness comes with greater simplicity, rather than greater complexity, due to more powerful parameterization;
- With such expressiveness, adding a new kind of pattern, either vertical or horizontal, to XML processing is just a programming task, rather than a language (re-)design task.

The rest of the paper is organized as follows. Section 2 reviews existing XML processing approaches, then the theory on which our approach is based. Section 3 introduces the use of patterns as first-class objects and the construction of complex patterns from simple ones, and shows how these patterns contribute to better type-safety and higher parameterization. Section 4 briefly shows how our approach extends to new kinds of complex patterns. Section 5 draws conclusion and discusses some open issues of our approach.

2 Related Work

2.1 XML Query Languages

In the early years of XML, special-purpose XML query languages such as Lorel (Abiteboul, Quass, McHugh, Widom & Wiener 1997), YATL (Cluet, Delobel, Siméon & Smaga 1998), XML-QL (Deutsch, Fernandez, Florescu, Levy & Suciu 1999), XQL (Robie, Lapp & Schach 1998) and XSLT (Clark 1999) were invented to handle query and transformation of XML data. They are typically untyped, handling both tag names and element content as strings.

These query languages have very limited programming power, unable to express sophisticated computations on XML data. In many settings, the queries and their results must be passed, at runtime, to other application programs for further processing. These transfers are usually in a low-level format such as strings, requiring extra programming effort and runtime overhead for parsing and type-checking. The type safety of XML manipulation programs then relies on type-checking at runtime by explicit checking code inserted by programmers at development time. The correctness and completeness of the checking code are not guaranteed.

XSLT uses XPath (Clark & DeRose 1999) expressions as search patterns. XPath is powerful at expressing a wide range of complex vertical patterns, but XSLT is limited in programming power, and incapable of sophisticated computation. The other languages are quite restricted both in expressing vertical patterns and in programming. None of them is able to express horizontal patterns systematically, although they can hardcode individual ones (e.g. sibling axes in XPath can represent simple horizontal patterns).

2.2 Native XML Processing

In recent years, attempts have been made to merge XML processing into general-purpose programming languages. Typical approaches use special types to represent XML data and special expressions for search patterns, in addition to regular programming language features. The most recent efforts include XJ (Harren, Raghavachari, Shmueli, Burke, Sarkar & Bordawekar 2004), XQuery (Boag, Chamberlin, Fernandez, Florescu, Robie & Simeon 2005) and Cω (Bierman, Meijer & Schulte 2004, Meijer, Schulte & Bierman 2003) focusing on vertical patterns, and XDuce(Hosoya & Pierce 2003) and CDuce(Benzaken, Castagna & Frisch 2003) focusing on horizontal patterns. In terms of programming style, XJ and Cω are object-oriented, while XQuery, XDuce and CDuce are functional.

XJ and XQuery enforce static typing against XML schemas rather than native types of the programming languages, and express search patterns using embedded strings; hence type mismatches still exist to some extent. Cω, XDuce and CDuce express XML data and search patterns fully in native mode with static typing, so that XML processing can be handled within a single language.

The inability to parameterize structures and patterns, and poor extensibility, are two common shortcomings in all these languages. First, these languages can only parameterize XML data items, not structures of these items, nor the patterns to match the structures, because the latter are not first-class entities. Traversal of heterogeneous XML structures has to rely on runtime type casts even if the XML data are parsed into well-typed form, and patterns have to be hard-coded in programs. Second, these languages only allow specific kinds of patterns and cannot be extended to other kinds easily in a type-safe way. An extension to a new kind of pattern requires new features to be added to the language; the type system has to be modified; and so does the compiler.

XJ extends Java with XML data types and XPath expressions, capable of handling vertical patterns conforming to XPath 1.0 (Clark & DeRose 1999). It expresses XML element types as Java classes, and uses special embedded strings containing XPath expressions as search patterns. Static typing of these XML types and embedded pattern strings against XML schemas is enforced by a special type checker. Because the type checking is against XML schema types, not native Java types, XML data and pattern expressions are not fully type-safe in Java. Since search patterns are just strings, there is a potential to include patterns other than XPath expressions, but only in an untyped way (or at best typed against XML schemas, not Java). Also, the special type checking requires that schemas for XML data are always available and trustworthy, which is unrealistic in many situations.

XQuery is designed as a query language but is equipped with some basic functional-programming features. It is intended to be a language for XML processing analogous to SQL for relational data processing. It aggregates many features from older XML query languages and SQL, and its data model and type system fully conform to XML and XML Schema specifications. It is a superset of XPath 2.0 (Berglund, Boag, Chamberlin, Fernndez, Kay, Robie & Simon 2005), making XPath expressions native, and so it is fully capable of handling vertical patterns of the XPath form. On the other hand, XQuery has only very limited functional programming features. Except in user-interactive settings, its expressions are supposed to be embedded in host programs in other languages for processing of query results. In such situation, the impedance mismatch problem still exists, just as in XJ, since XQuery is only typed in terms of XML schemas. The mismatch between XML schema types and host-language types weakens the safety of XML processing programs. The only advantage over XJ is that, in the absence of XML schemas, XQuery expressions can still be type-checked to some extent based on the type information in the expressions themselves.

Cω is intended to extend C#, another general-purpose programming language, with native types that support both object-oriented, relational and semi-structured data models, so that it can unify the processing of all these kinds of data. It introduces three new kinds of types: stream, anonymous struct and choice, roughly equivalent to list, heterogeneous tuple and sum types in functional languages. It uses the notion of content class for expression of XML schemas. For example, suppose an XML schema for geographical data has a country element type, with name, population and zero or more provinces as child elements. It then can be encoded as a content class `Country` as:

```
class Country {
    struct{ string name; float population; Province* provs; };
    ...  // appropriate constructor
    void increasePopulation(float percentage){...}
    ...
}
```

which contains an anonymous struct holding `name`, `population` and a stream of `Province`. In turn, `Province` is another content class (declaration not shown here) for province element type, which may have children name, population and a stream of `City`, and so on. Suppose `canada` is an instance of `Country`. The pattern to get the population of Canada can then be expressed as `canada.population`. To accommodate XPath-style vertical patterns, Cω also introduces filter expressions such as `Country[name=="Canada"]`, and transitive query expressions such as `Country...population` for population data appearing at arbitrary depth below countries. For example, the following method returns a stream of populations of cities in a given country:

```
virtual float* getPopulation(Country c1) {
    foreach (p in c1...City.population) yield return p;
}
```

The expressiveness of Cω for patterns in XML processing is roughly equivalent to XPath 1.0 (Clark & DeRose 1999) without backward axes. In contrast to XJ and XQuery, XML data and pattern expressions in Cω are fully native, expressed by identifiers all having Cω native types. There is no impedance mismatch problem. However, the pattern to search, such as `c1...City.population` in the above method, has to be hardcoded in the program and cannot be passed to a method parameter in a typed manner, so that it is not possible to have a general method to search for user-defined target data, something like `get(somePatternType pattern, Country c1)`. Moreover, adding other kinds of patterns, for example XPath backward axes, self-nested structures, or horizontal patterns would require large changes to the language.

XDuce and CDuce are functional-programming languages with regular-expression types added to general-purpose functional language features. These two languages use regular expressions to denote XML element types, and to define horizontal patterns to match the elements. For example, the country element type above can be declared in CDuce as:

```
type Country = <country>[Name Population (Province)*]
type Province = <prov>[Name Population (City)*]
type City = <city>[Name Population ...]
type Name = <name>[String]
type Population = <pop>[Int]
```

Traditional Pattern matching can be used to locate all population items and make some update to them in a piece of XML data:

```
let updatePop (x:<_>[_*]) :<_>[_*] =
    let [ y ] =
        xtransform [ x ] with
            <pop>[(z & Int)] -> [ <pop>[(z*101/100)] ]
    in y
```

This CDuce function uses regular-expression type `<_>[_*]`, meaning an element with any tag name and any content, to constrain both the parameter and result, and regular-expression type `<pop>[Int]`, meaning an element with tag name "pop" and an integer as content, to match target items for update. It traverses the whole structure of a given piece of XML data `x` using the macro iterative operator `xtransform`, matches any population element and increases it by 1%.

In XDuce and CDuce programs, patterns are well-typed and handled natively. CDuce can even encode XPath-like vertical patterns with child axes (though not descendant axes). However, just as for Cω, search patterns such as `<pop>[Int]` in the above CDuce program are not first-class terms and cannot be referenced and passed as well-typed parameters. And new kinds of patterns are not easy to include without significant extensions to the languages.

2.3 bondi and Pattern Calculus

bondi (Jay 2004*a*) is a general-purpose functional programming language designed to allow many forms of genericity. Instead of aggregating features for XML data processing found in the existing wide variety of XML query and transformation languages, bondi has a very general extension to functional language features to achieve a higher degree of modularity and program re-use.

The extension is based on a sound theory, the *Pattern Calculus* (Jay 2004*c*, Jay 2004*b*, Jay 2004*d*), which:

- treats structures and patterns as first-class objects with equal importance to data and functions, allowing them to be referenced and passed as parameters, achieving parameterization of structures, access paths and search patterns;
- allows a generalized form of pattern matching, without requiring the pattern cases to be the same type.

Hence, in bondi, control flow can be determined by structures, not just datum values; and structures and structure-matching patterns are natively well typed, can be used as values, passed around as parameters, and matched in a general way.

In the same way that data and function parameterization make data and function polymorphism possible, the treatment of structures and patterns and the generalization of pattern matching in bondi make possible three new forms of polymorphism: structure polymorphism, path polymorphism and pattern polymorphism. They provide new expressive power and create the opportunity to represent XML processing in a well-typed, highly parametric and highly extensible way. The next section will explain these forms of polymorphism and how they can be used in XML processing.

3 Parameterizing Structures and Patterns

Programming (and maintenance) are simpler when programs are built so that as much of their behavior is captured by parameters as possible. Often this has a secondary benefit that the resulting program is simpler and easier to understand (many of the cases have become different parameter choices). Programming languages that support the passing of data and functions as parameters (higher-order functions) or use subtyping to pass objects of varying behavior are plentiful, but until the Pattern Calculus (Jay 2004*c*, Jay 2004*b*, Jay 2004*d*) there has not been general account of how to pass around information about structures, and patterns to match these structures within a typed programming language. The Pattern Calculus, and its implementing language bondi, support all these kinds of parameter passing, achieving polymorphism on data, functions, subtypes, structures, paths and patterns within one typed programming language. The latter three, achieved by parameterizing structures and patterns, are particularly suited to describe XML access paths, and can greatly simplify programming for XML manipulation. This section explains these three new forms of polymorphism by introducing a sequence of XML processing examples requiring deep parameterization, and shows how simple and type-safe it is to design highly-parametric functions for XML data processing.

3.1 A Motivating Scenario

Suppose we have a data repository containing geographical information and we want to carry out the following operation: *Add 1% to the population of all of Canadian cities.* How could we express such an operation?

The first way is what might be called assembly language programming: a specific program that traverses the structure in the repository, finds all of the places where Canadian cities are present, and then finds their population elements and adds 1% to them. The problem is that if we decide to change the problem in any way we have to rewrite and recompile the program.

All high-level programming languages allow the amount by which the populations are to be incremented to be extracted and expressed as a parameter. So we might write something like:

$$IncrementPopsofCanadianCities(1\%)$$

This small change increases the generality of the program in the sense that we can make many different changes without rewriting or recompiling the program. The program is generic with respect to one argument.

Many programming languages also allow us to make the operation that is to be done to the populations of Canadian cities into a parameter as well. So we might write:

$$UpdatePopsofCanadianCities(incrementby, 1\%)$$

Now it is trivial to decrement the populations instead.

The next level of generality is to make the parts of the structure where the function is applied into a parameter as well. So we might write:

$$updateCanadianCities(Pops, incrementby, 1\%)$$

Now it is trivial to increment (or decrement) cities' *areas* instead of their populations. Most query languages, either for databases or for semistructured data, are powerful enough to allow this kind of programming, but many general-purpose languages have trouble because the contexts that define the regions where the function is to be applied are constructed in different ways and look different to the type system.

A further extension is to make the particular units within Canada that are being considered into a parameter. So we might write:

$$updateInCanada(City, Pops, incrementby, 1\%)$$

Now the program is generic in the pattern that describes *where* the increment is to be applied (cities above populations). It will work regardless of whether cities are immediately below countries, e.g., capitals such as Ottawa or Washington D.C., or accessed via intermediate layers such as states or provinces.

Now let us parameterize on the country too:

$$update(CountryName == \text{“}Canada\text{”}, City, Pops, incrementby, 1\%)$$

The code involves a side-condition to check on a related structure.

Now we see that the parameters *“Canada”*, *City* and *Pops* are all related and it is the *connections* between them that define the real parameter of interest. So we could rewrite the code as:

$$update(Canadian\_City\_Pop, incrementby, 1\%)$$

which has a (complex) pattern parameter. Now if we want to search for more complicated structures within the geographical database, we don't have to keep building more complicated functions; rather, the complexity is expressed in the choice of a complex pattern *parameter* of the standard *update* function.

This example shows the many levels of need for genericity in processing semi-structured data. Most programming languages and query languages can satisfy some of these needs, but the following subsection will show that bondi is the first to handle them all in one language, and in a natural way.

3.2 Parameterizing in bondi

This subsection encodes the examples above in bondi; they have all been executed and also appear in the file "xmldata.bon" at the bondi web-site (Jay 2004*a*). Language features will be explained as they are used without attempting a full introduction here. As a convention, `a, b, c, d, ...` are used as variables for types and ..., w, x, y, z are variables for values.

Define a datatype of populations by

```
datatype popul = Pop of float;;
(* unit: thousand people *)
```

This declaration introduces both a new type `popul` and, a new term, its constructor `Pop` of type `float->popul`. We can define a function for updating populations by pattern-matching:

```
let (atPopIncrementBy1Percent:popul->popul) x =
    match x with
    | Pop z -> Pop (z * 1.01);;
```

When applied to a term of the form `Pop x` it returns `Pop (x*1.01)`. This function can be parameterized with respect to the action to be taken by defining

```
let (atPopApply:(float->float)->popul->popul) f x =
    match x with
    | Pop z -> Pop (f z);;
let incrementBy1Percent x = x*1.01;;
let atPopIncrementBy1Percent = atPopApply incrementBy1Percent;;
```

Evaluation of the new version of `atPopIncrementBy1Percent` reduces to the old one by substituting for the variable `f`.

More generally, we can consider increasing populations stored in larger data structures, e.g., lists defined by

```
datatype list a =
    | Nil
    | Cons of a and list a;;
```

This example defines a data type `list` which takes one parameter, a, which is the type of the list elements. It has two constructors: `Nil` which builds an empty list and `Cons` which constructs a new list from an element and a (sub)list. We use `[x, y, z, ...]` as syntax sugar for `(Cons x (Cons y (Cons z ...)))` and `[ ]` for the empty list `Nil`.

The function

```
let (listMap:  (a->b) -> list a -> list b) f x =
    match x with
    | Nil -> Nil
    | Cons y z -> Cons (f y)(listMap f z);;
```

takes a function `f` as its first argument and applies it to every element of the second argument, a list. `listMap` is defined by pattern-matching over the two list constructors. For example,

```
listMap incrementBy1Percent
```

acts on lists of floats and

```
listMap atPopIncrementBy1Percent
```

acts on lists of populations. This illustrates how `listMap` is polymorphic in the choice of types a and b that represent the list entries, i.e., `listMap` is *data polymorphic*.

Of course, populations may appear as data in all sorts of structures, not just lists. This situation can be handled using a mapping function that is parametric in the choice of *structure* type as well as in the choice of the *data* types, i.e., function

```
map1:  (a->b) -> c a -> c b
```

whose type includes a type variable `c` representing the structure, e.g., `list`. We say function `map1` is *structure polymorphic*. The definition of `map1` is more complex than its type suggests as it relies on the theory of data structures developed in (Jay 2004*c*).

Even `map1`, however, is not flexible enough for our purposes, since a typical database is not going to be as homogeneous as type `(c popul)`, having only one type of elements. There is no reason to single out populations while ignoring, say, city names and areas.

Instead, let us define a function that acts on populations wherever they occur, by

```
let (updatePops:(float->float)->d->d) f x =
    match x with
    | Pop z -> Pop (f z)
    | y z -> (updatePops f y) (updatePops f z)
    | z -> z;;
```

Note that the patterns of three matching cases are of different types. This generalized form of pattern matching is allowed by Pattern Calculus with a less-restricted typing requirement (Jay 2004*c*). The first case is the same as `atPop` but the second and third cases cause the action to be propagated to all parts of the data structure. That is, the pattern `y z` matches against any compound data structure (e.g., `Cons s t`), and causes both parts of the compound (e.g., `Cons s` and `t`) to be updated, while the final case is used to terminate at atoms of data. They can match different type of structure in each recursive call. For example,

```
updatePops incrementBy1Percent [Pop x1, Pop x2]
```

evaluates to `[Pop x1*1.01,Pop x2*1.01]`; but

```
updatePops incrementBy1Percent ([Pop x1],Pop x2)
```

evaluates to `([Pop x1*1.01],Pop x2*1.01)` even though the populations appear on different levels of the data structure. Thus `updatePops` is *path polymorphic* since it can adapt to different data access paths.

Examining the program above, it is clear that the constructor `Pop` is playing a completely passive role, and so is ripe for parameterization. Define

```
let (update:lin(a->b)->(a->a)->d->d) \P f x =
    match x with
    | P z -> P (f z)
    | y z -> (update P f y) (update P f z)
    | z -> z;;
```

so that function `updatePop` can now be defined by `update Pop`.

The program `update` arises naturally from our earlier examples, but has a number of unusual technical features. First some conventions: capitalized variables such as P are always free unless explicitly bound as in \P (to be thought of as λP). Thus, the pattern `P z` contains a free variable P and a binding variable z. Evaluation of `update Pop` will substitute `Pop` for P so that the pattern above becomes `Pop z`. That is, `update` is *pattern polymorphic* since it takes a parameter used to build patterns.

Some care is required when substituting into patterns, so such variables are required to be *linear* as indicated by the linear type `lin(a->b)`, meaning that the function of type `a->b` uses its argument exactly once. Linear terms are explained in detail in (Jay 2004*b*). For this paper, we will pretend that all linear terms are constructors though there are important alternatives. So for now `lin(a->b)` is the type of a constructor with an argument of type `a` for a data structure of type `b`. For example, `Pop` has type `lin(float->popul)`.

Similarly, we can define a function `check` that simply checks that some property holds for some argument of the given constructor, by:

```
let (check:lin(a->b)->(a->bool)->d->bool) \P f x =
    match x with
    | P z -> f z
    | y z -> (check P f y) || (check P f z)
    | z -> False;;
```

where `True, False` are two constant constructors of type `bool` as usual and `||` is logical-or.

Suppose now that the goal is to update the populations of only cities, while leaving other populations unchanged. For example, consider XML geographical data conforming to the schema:

```
<xs:element name="cityname" type="xs:string"/>
<xs:element name="popul" type="xs:decimal"/>
    <!-- unit:  thousand people -->
<xs:element name="river" type="xs:string"/>

<xs:element name="city">
    <xs:complexType><xs:sequence>
        <xs:element ref="cityname"/>
        <xs:element ref="popul"/>
        <xs:element ref="river" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xs:sequence></xs:complexType>
</xs:element>

<xs:element name="provname" type="xs:string"/>
<xs:element name="province">
    <xs:complexType><xs:sequence>
        <xs:element ref="provname"/>
        <xs:element ref="popul"/>
        <xs:element ref="city" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xs:sequence></xs:complexType>
</xs:element>

<xs:element name="countryname" type="xs:string"/>
<xs:element name="country">
    <xs:complexType><xs:sequence>
        <xs:element ref="countryname"/>
        <xs:element ref="popul"/>
        <xs:element ref="province" minOccurs="0"
                    maxOccurs="unbounded"/>
    </xs:sequence></xs:complexType>
</xs:element>
```

In an implementation of our approach, a validating XML parser is needed to transform XML data into bondidata format for processing. Assuming such parsing, the above schema can be denoted as bondidata structures:

```
datatype cityname = CityName of string;;
datatype popul = Pop of float;;
(* unit: thousand people *)
datatype river = River of string;;
datatype city = City of cityname * popul * list river;;

datatype provname = ProvName of string;;
datatype province = Prov of provname * popul * list city;;

datatype countryname=CountryName of string;;
datatype country = Country of
          countryname * popul * list province;;
```

Here ∗ represents product type with the usual functional programming convention, and constructor `Pair` of `a` and `b` represents pairing data items. `(x, y)` is syntactic sugar for `Pair x y`, and tuple `(x, y, z, ...)` is nested pairs. For programming convenience, we always encode children of an XML element as nested pairs as in the above declarations, e.g. a city element for Kingston are encoded as:

```
City("Kingston",Pop 100.0,["St.Lawrence River"])
```

Now applying `update Pop f` to a piece of geographical data will act on all of the city, province and country populations indiscriminately. However, the function

```
update City (update Pop f)
```

gives the desired behavior. Although correct, this is not quite satisfactory, since it requires two updates. More complicated access patterns typical of XML will then require three or more updates, and there is still the challenge of checking side-conditions, e.g., that the city is in Canada.

The solution is to construct an abstract structure type for the complex patterns that represents all of the information about how to access the data items. In simple cases this will be given by a complete hierarchical path, but in general the access information will be partial. For example, it is not necessary to know everything above a city to update its population, and most information along the path down to that city is not interesting. Let us call such a partial description of a path a *signpost* since it guides the way to target data items.

For the purposes of encoding the motivating examples, let us consider three sorts of signposts. It will be easy to add more sorts as needed. A *goal* is a constructor whose argument is the singular pattern of the target item of interest. A *stage* is a constructor that constructs a signpost from a leading simple pattern of the path and the rest of the path. A *detour* is a path that has a side-path with a filtering condition to check before continuing on the main path. Thus we obtain a structure:

```
datatype signPost
  at a b c =
  |Goal of lin(c->b)
  at (a1,a2) (b1,b2) c =
  |Stage of lin (a1->b1) and signPost a2 b2 c
  |Detour of detourPath a1 b1 and signPost a2 b2 c;;
```

Here "`at`" indicates pattern matching with different forms of the type arguments. `signPost` takes three type arguments, the first two of which can be pairs of types. `detourPath` is a helper structure to represent a filtering condition in a `signPost`:

```
datatype detourPath
    at a b =
    | DetourGoal of lin(a->b) and (a->bool)
    at (a1,a2) (b1,b2) =
    | DetourStage of lin(a1->b1) and detourPath a2 b2;;
```

`signPost` and `detourPath` are similar in form to data structures such as `list`, but they are not data structures any more because they contain pattern type `lin(a->b)`, which is not data type. We call them *pattern structures*.

Note that in `signPost` an extra parameter type `c` is used to expose the content type of the final element, the goal, for programming convenience. It enables general programming of computation with such paths as parameters. For example, to search for all populations with the partial path `(...country...city...popul)`, the compound pattern can be encoded as:

```
let popPath1 = Stage Country (Stage City (Goal Pop));;
```

and to search for all populations of Canadian cities, we use `Detour`:

```
let dpath = DetourGoal CountryName ((==) "Canada");;
let popPath2 = Stage Country
                    (Detour dpath (Stage City (Goal Pop)));;
```

Note that `(==)` is a boolean function of type `a->b->bool`, which takes two arguments, so that `((==) "Canada")` is a boolean function of type `a->bool`.

Now function `check` can be modified to act on `detourPath`, and `update` be modified to act on `signPost`, as follows.

```
let (checkd:(detourPath a b)->d->bool) p x =
  match p with
  | DetourGoal \P f -> check P f x
  | DetourStage \P p1 -> check P (checkd p1) x;;

let (updates:(signPost a b c)->(c->c)->d->d ) s f x =
  match s with
  | Goal \P -> update P f x
  | Stage \P s1 -> update P (updates s1 f) x
  | Detour dp1 s1 ->
          if (checkd dp1 x)  (* the detour *)
              then updates s1 f x
          else x;;
```

Note that function `updates` ("s" stands for signpost) invokes the simple version `update` for singular patterns. It uses pattern matching to explore the structure of a given path pattern, that is, a `signPost`. If the path pattern is a singular pattern, `update` is invoked directly. If the path pattern is a `Stage`, `update` is used to search for the preceding singular pattern, then from the matching points the search for the rest of the path pattern continues. `checkd` also acts in a similar way.

If the path pattern given to function `updates` is a `Detour`, the function checks whether the detour path got a match and whether the content of the match satisfies the carried boolean function. If so, the function goes back to the starting point and continue the search for the rest of the main path pattern. If the detour does not get a match or the carried boolean function fails, the function returns unchanged data. Now it is straightforward to increment all populations of all Canadian cities, if `data` is the data repository containing geographical information:

```
updates popPath2 incrementBy1Percent data
```

Note that this executes independently of the presence of provinces.

3.3 Folding

Given bondi's support for parameterization over data structures and data access patterns, we can design other general functions in much the same way as `map1`, `update` and `updates`. In this subsection we define functions for the folding operation, which is the basis of many common operations on heterogeneous data structures. We also show by an example the simplicity of using the folding functions in XML data processing.

A function `foldleft1` can be defined (Jay 2004*c*), similar to `map1`, as:

```
foldleft1:  (a->b->a) -> a -> c b -> a
```

It traverses a homomorphic structure of type `(c b)` with all elements being only one type `b`, applying a given function to the values of all elements it finds to modify the given value of type `a`. For example, given a definition of integer addition function `add`, the application `foldleft1 add 0 AListOfInt` produces the sum of all integers in a list of type `(list int)`.

In the same way that `update` handles singular patterns appearing in various contexts of arbitrary heterogeneous data structures, a generalized `foldleftp` for heterogeneous structures can be defined, again simply using three-case pattern matching:

```
let (foldleftp:lin(a->b)->(e->a->e)->e->d->e) \P f x w =
    match w with
    | P z -> f x z
    | y z -> foldleftp P f (foldleftp P f x y) z
    | z -> x;;
```

A more sophisticated version that folds elements satisfying a complex path pattern (`signPost`) looks like this:

```
let (foldlefts:(signPost a b c)->(e->c->e)->e->d->e) s f x w =
    match s with
    | Goal \P -> foldleftp P f x w
    | Stage \P s1 -> foldleftp P (foldlefts s1 f) x w
    | Detour dp1 s1 ->
          if (checkd dp1 w)
              then foldlefts s1 f x w
          else x;;
```

Many essential XML processing operations can be expressed as using `foldleftp` and `foldlefts`. For example, extracting information from XML data based on a search pattern and a filter is the most common kind of XML query. It can be easily implemented by `foldlefts`.

Suppose we want a list of names of cities whose population is bigger than 300 (in units of thousands). This query consists of three components: the pattern to search for: `...city...cityname`; the data filter: population > 300; and the way to construct the result. The search pattern is easy to describe by an instance of a `signPost`, and the filter is a boolean function carried by a `Detour` pattern:

```
let dpath = DetourGoal Pop ((>) 300.0);;
let namePath = Stage City (Detour dpath (Goal CityName));;
```

Collecting matching items into a final result is a `foldlefts` operation in bondi. An accumulating function will be given to the folding function as a parameter, to accumulate matching items. Users can use different accumulating functions for different ways of constructing the final result. If the result is to be a list of strings for city names, i.e., of type `list string`, the accumulating function can be as simple as:

```
let (listInsert:  list a -> a -> list a) x y =
    match x with
    | Nil -> Cons y Nil
    | Cons z w -> Cons y (Cons z w);;
```

Of course more sophisticated accumulating functions can be designed, for example to check for duplicates, or to construct results into a structure other than a flattened string list.

Given the pattern and accumulating function, the task is straightforward (again, `data` is the geographical data repository):

```
let nl = foldlefts namePath listInsert [ ] data;;
```

Many other essential XML processing operations, such as extraction while preserving or restructuring original structures, indexing and sorting, are basically folding operations as well and can be implemented in a similar way using the folding functions. More examples are available in one of our earlier reports (Huang, Jay & Skillicorn 2005*b*).

Designing highly-parametric general functions in bondi, such as `update`, `updates`, `foldleftp` and `foldlefts`, is as simple as pattern-matching several cases. Such simple functions allow us to perform a large class of XML search and transformation operations within a general-purpose programming environment easily. These functions can be formalized as library components of bondi. The only task left for users is to map complex search patterns into instances of appropriate pattern structures such as `signPost`, and this may also be automated.

4 More Complex Patterns

Besides designing new generic functions, another way to extend XML processing capability is to include more kinds of patterns. In the previous section we defined a pattern structure, `signPost`, which is able to express a large class of common vertical patterns. To handle more complex patterns, we can add more constructors to `signPost`, or even define new pattern structures as appropriate. In bondi this is a programming task, in contrast to other existing XML processing approaches where new kinds of patterns need new language features at best, and are impossible at worst.

We have experimented how to extend our approach to handle complex patterns in XPath style, vertical regular-expression style, and horizontal regular-expression style. These patterns have been considered individually in other existing approaches but have never appeared fully together in one language. Our extensions for these new patterns only need declarations of new pattern structures and changes to programs processing data using these structures. None of our extensions require any changes to the language bondi itself.

For example, we can declare a pattern structure to represent regular expressions:

```
datatype regexp
    at a b =
    | Single of lin(a->b)
    | Kstar of lin(a->b)
    at (a1,a2)(b1,b2)
    | Concat of regexp a1 b1 and regexp a2 b2
    | Altern of regexp a1 b1 and regexp a2 b2;;
```

and use this structure to encode patterns of horizontal regular-expression style. We can design functions for search, update and folding of target data matching such patterns.

Further details about handling complex patterns in XPath style, vertical regular-expression style and horizontal regular-expression style can be found in the report (Huang, Jay & Skillicorn 2005*a*).

5 Conclusion

The strongly-typed general-purpose programming language bondi, based on Pattern Calculus, treats structures and patterns as first-class objects, and allows a generalized form of pattern matching with less restricted typing rules. These increases in expressiveness create new forms of polymorphism, especially path polymorphism and pattern polymorphism. Path polymorphism enables traversal of data with heterogeneous structures, automatically adapting to different data-access paths on the fly in a well-typed manner. Pattern polymorphism allows data-access patterns to be passed as well-typed parameters, and be composed into complex pattern structures.

With the new expressive power, we have shown that we can define general programs using generalized pattern matching and parameterizing structures and patterns, implementing a large class of essential XML processing operations. Compared with those from other existing XML processing approaches, bondi programs are simpler and more modular due to higher parameterization and more freedom for pattern matching. These programs are also safer because static typing is enforced not only on data items and functions, but also on structures and patterns.

With the new expressive power, we have also shown that we can easily create new pattern structures or expand existing ones to handle new kinds of complex patterns in XML manipulation. These pattern structures can be treated as freely as data structures. They can be constructed, pattern-matched, traversed at runtime, and passed as values to parameters, making programming with them very flexible and simple. They carry all necessary type information, enabling static type verification for the programs that use them. Extensions to new kinds of patterns require only programming not, as in the other existing approaches, language design or revision. This makes our approach highly extensible, and applicable for a richer set of complex patterns than other XML query and transformation languages.

Given that our approach manipulates XML within one programming language with simplicity, strong type-safety and high extensibility, it is easy to integrate back-end data-access programming with front-end user-interface programming in a single system. The approach thus represents the first steps to solving the impedance mismatch problem.

Of course, there is still a long way to go to use this approach in practical applications. A lot of implementation effort is required and some issues are still open for further investigation.

- It is not expected that XML data users has to do the bondi programming. They will even not need to know about patterns and pattern structures. A library for common XML computations such as search, update and folding can be built. Pattern structure declarations and constructions could be automated, and XML- and XPath-style expressions could be adopted as syntax sugar.
- Currently only an interpreter for bondi is available. Given the high level of language abstraction and polymorphism, it is desirable to compile bondi programs into a format that can be optimized for performance.
- Examples given in this paper assume that XML data are transformed into bondi data structures in memory. When facing large-scale data, although bondi data structures could also be stored in external repositories, it is not yet clear how to optimize the data access for the performance of the repositories.

References

Abiteboul, S., Quass, D., McHugh, J., Widom, J. & Wiener, J. (1997), 'The Lorel query language for semistructured data', *Int. J. on Digital Libraries* **1**(1), 68–88.

Bancilhon, F. & Maier, D. (1988), Multi-language object-oriented systems: New answers to old database problems, *in* K. Fuchi & L. Kott, eds, 'Future Generation Computers II', Amsterdam, North-Holland.

Benzaken, V., Castagna, G. & Frisch, A. (2003), Cduce: an xml-centric general-purpose language, *in* 'Proc. of 2003 ACM SIGPLAN Int. Conf. on Functional Programming', ACM Press.

Berglund, A., Boag, S., Chamberlin, D., Fernndez, M., Kay, M., Robie, J. & Simon, J. (2005), 'Xml path language (xpath) 2.0 - w3c working draft'. `www.w3.org/TR/2005/WD-xpath20-20050211/`.

Bierman, G., Meijer, E. & Schulte, W. (2004), 'The essence of data access in cω'. `research.microsoft.com/~emeijer/Papers/popl.pdf`.

Boag, S., Chamberlin, D., Fernandez, M., Florescu, D., Robie, J. & Simeon, J. (2005), 'Xquery 1.0: An xml query language - w3c working draft'.

Clark, J. (1999), 'Xsl transformation(xslt): Version 1.0 - w3c recommendation'.

Clark, J. & DeRose, S. (1999), 'Xml path language (xpath): Version 1.0 - w3c recommendation'. `www.w3.org/TR/xpath`.

Cluet, S., Delobel, C., Siméon, J. & Smaga, K. (1998), Your mediators need data conversion!, *in* 'ACM SIGMOD International Conference on Management of Data', Seattle, Washington, USA, pp. 177–188.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A. & Suciu, D. (1999), 'A query language for XML', *Computer Networks* **31**(11–16), 1155–1169.

Harren, M., Raghavachari, B., Shmueli, O., Burke, M., Sarkar, V. & Bordawekar, R. (2004), XJ: Integration of XML processing into Java, *in* 'Proc. WWW2004', New York, NY, USA.

Hosoya, H. & Pierce, B. (2003), 'Xduce: A typed XML processing language', *ACM Transactions on Internet Technology* **3**(2), 117–148.

Huang, F. Y., Jay, C. B. & Skillicorn, D. B. (2005*a*), Dealing with complex patterns in XML processing, Technical Report 2005-497, School of Computing, Queen's University. `www.cs.queensu.ca/TechReports/Reports/2005-497.pdf`.

Huang, F. Y., Jay, C. B. & Skillicorn, D. B. (2005*b*), Programming with heterogeneous structure: Manipulating XML data using bondi, Technical Report 2005-494, School of Computing, Queen's University. `www.cs.queensu.ca/TechReports/Reports/2005-494.pdf`.

Jay, C. B. (2004*a*), 'bondi web-page'. `www-staff.it.uts.edu.au/~cbj/bondi`.

Jay, C. B. (2004*b*), 'Higher-order patterns'. `www-staff.it.uts.edu.au/~cbj/Publications/higherorderpatterns.pdf`.

Jay, C. B. (2004*c*), 'The pattern calculus', *ACM Trans. Program. Lang. Syst.* **26**(6), 911–937.

Jay, C. B. (2004*d*), 'Unifiable subtyping'. `www-staff.it.uts.edu.au/~cbj/Publications/unifablesubtyping.pdf`.

Meijer, E., Schulte, W. & Bierman, G. (2003), Unifying tables, objects and documents, *in* 'Proc. DP-COOL 2003', Uppsala, Sweden.

Robie, J., Lapp, J. & Schach, D. (1998), 'Xml query language (XQL)'. `www.w3.org/TandS/QL/QL98/pp/xql.html`.

Wadler, P. (2004), 'Links'. `homepages.inf.ed.ac.uk/wadler/papers/links/links-blurb.pdf`.

A Relational Account of Objects

Clara Murdaca & C. Barry Jay
University of Technology, Sydney
{murdaca,cbj}@it.uts.edu.au

Abstract

A relational account of objects provides a single unifying data model for both object-oriented programming languages and relational databases. Type variables used to represent unknown fields in the programming language correspond to discriminators in relations.

1 Introduction

Relational databases have proved their worth in storing and manipulating large amounts of data, while object-oriented programming languages are excellent at organising computation in a way that is flexible and maintainable. Ideally, object-oriented programs should be compiled to take full advantage of data stored in relational tables but this is difficult, if not impossible, while ever the data model for classes differs from that for relations. This paper proposes a relational account of objects suitable for both programming and the organisation of relational databases.

Various approaches to storing objects have been considered. Simplest is to store the objects in the database just as they are stored in working memory of the language, to create an *object base* (see e.g. (Bloom & Zdonik 1987, Breazu-Tannen, Buneman & Ohori 1991, Atkinson, Bancilhon, DeWitt, Dittrich, Maier & Zdonik 1994, Leontiev, Ozsu & Szafron 2002)). Alternatively, one can aim for *persistence* (Agrawal, Dar & Gehani 1993, Richardson, Carey & Schuh 1989), or attempt a mixture of data models, as in C-omega (Bierman, Meijer & Schulte 2005). Unfortunately, these all reduce the time and space efficiency of the store in ways which have proved resistant to improvement. An emerging approach (described in Section 2) is to try and model the classes as relations, with objects modeled as rows in tables. This is non-trivial since the standard model of classes is quite different from that of relations. In particular, relations do not support a natural interpretation of sub-classes, resulting in a variety of competing approaches; these can be compared in terms of the efficiency with which objects can be stored and retrieved, and the ease of maintenance.

However, none of these approaches to storage is able to combine efficient data storage and retrieval with the modular compilation of classes. Modular compilation is highly desirable when creating large systems. It is also a pre-condition for being able to invest in significant optimsation of code to take advantage of the efficiency of the underlying query language. The typical approach has an object-oriented process request an object from the database using some tool, updates the object and then uses the tool to return it to the database. With a shared object model and modular compilation it may be possible to compile some methods directly into queries.

One consequence of modular compilation is that the introduction of new sub-classes should not change the representation of their super-classes, e.g. should not require re-factorisation. The simplest way to achieve this is to require that each class have its own relation.

Here then are four desirable properties that such a model should have.

(P1) Data storage is efficient.

(P2) Compilation is modular.

(P3) There is one relation per class.

(P4) Compiled methods take advantage of queries.

Object bases and systems with persistance do not have the first property, so let us focus on modeling classes as relations, as described in (Ambler 2003). Only one of the approaches therein satisfies (P3); in the other three approaches the relation(s) describing a class evolve as new sub-classes are created. However, all of the approaches invoke a method by converting a row into an object and proceeding in the usual object-oriented style: determine its class and execute the corresponding class code. To adopt this approach when compiling into queries would be to import the whole notion of class into the query language, rather than exploit the inherent nature of queries.

More suitable for the purpose is an alternative approach to object-orientation, outlined in (Jay 2004*b*, Jay 2004*d*). Objects are represented by data structures built from their fields, while methods are given as functions that use pattern-matching to identify the appropriate algorithm. Sub-classing and sub-typing are handled by using type variables to represent unknown fields, in an approach similar to, but not identical, to that of *row variables* (Remy 1989). This makes it easy to satisfy properties (P1) and (P3). The sub-type relationship is captured using *discriminators* in relations. Matching on constructors in the programming language becomes a query on the discriminator, making it easier to represent methods as queries (P4). Finally, separate compilation (P2) can be supported by treating method specialisation as the addition of a new case to an existing (compiled) pattern-matching function.

The elaboration of this approach is ongoing work. The basic theory is supplied by the *pattern calculus* (Jay 2004*c*) which has been implemented in the programming language **bondi** (pronounced "bond-eye") (Jay 2004*a*) which is able to support classes and subtyping. These can be used to create relations in a relational database so that objects can be created, stored and retrieved in a natural way. Class compilation is already modular in **bondi** but does not yet exploit the power of the query language.

This paper introduces the *discriminating object model* and shows how it satisfies properties (P1–P3). It underpins both the class model of **bondi** and its organisation of relational data, which can then be used to illustrate the ideas. This provides a foundation for exploring (P4).

The structure of the rest of the paper is as follows. Section 2 reviews current object relational mapping techniques. Section 3 reviews the pattern calculus. Section 4 presents our object model. Section 5 draws conclusions and considers future work.

2 Object-Relational Mapping

```
public abstract class Entity
       { private int entity_id;
         private String name;
         private String address;
       }

public class Student extends Entity
       { private String course;
       }

public class Employee extends Entity
       { private double salary;
         public double SalaryIncrease()
         { return salary * 1.10; }
       }

public class Casual extends Employee
       { private double hours;
         private double SalaryIncrease()
         { return salary * 1.06; }
       }
```

Figure 1: Running example in Java

There are various ways in which objects can be mapped to relational tables (Fussell 1997, Keller 1998). These can be classified into four main object relational mapping techniques according to their treatment of class hierarchies (Ambler 2003).

The example given in Figure 1 will be used to illustrate these four mapping techniques. Figure 1 defines a class hierarchy using Java syntax. It contains an abstract Entity class with subclasses Student and Employee and a Casual subclass of Employee. Both the Employee and Casual classes have a salaryIncrease method. A 10% increase salary is given to ordinary employees while casual employees are given a 6% increase in salary.

Let us now look at how each of the mapping techniques would map the classes defined in Figure 1 to tables in a relational database.

ORM1 All the subclass attributes of a root (super) class are stored in a single table. As given in Figure 2 the attributes of the Employee, Student and Casual classes are combined into a single table, called the Entity table. In this mapping technique the fields that are not relevant to particular row entry will be populated with Null values. For example, an ordinary Student will have a Null entry for the salary and casual fields corresponding to the Employee and Casual classes respectively. Unfortunately if there are many sub-classes then most fields will be Null. Also, each time a new subclass is defined the existing table needs to be restructured which makes database maintenance expensive. Note that although property (P4) can be satisfied, properties (P1), (P2) and (P3) will fail.

Entity
entity_id
name
address
course
salary
hours

Figure 2: ORM1: One table per root class

ORM2 One table is defined per concrete class but abstract classes do not correspond to a single table. Property (P3) holds for concrete classes and there is no need to restructure existing tables when a new sub-class is defined. However, all abstract class attributes must be duplicated in the relation of each concrete subclass so (P1) will fail. For example, in Figure 3 the fields of the abstract Entity class form part of the table mappings for the Student, Employee and Casual relational tables. In this technique the creation of a new subclass does not require restructuring of existing tables. However, it is not clear how to compile methods into queries, so (P4) is in doubt. For example, to compile the salaryIncrease method for employees requires being able to determine which employees in the table are ordinary employees and which are casual employees.

ORM3 A single table is defined for each class, regardless of whether it is an abstract or a concrete class, as illustrated in Figure 4. A single identification key, given as the entity_id is used as both a primary and foreign key to link the tables to replicate the class hierarchy.

Now (P1) and (P3) is satisfied, and additions to the class hierarchy do not require the restructuring of existing tables. Also, (P4) can be satisfied as the Employee and Casual classes are linked by the entity_id field. However, in order to establish whether an employee is casual it is necessary to query the table for casual employees. This may prove expensive, especially if there are many sub-classes to consider. So, (P3) will only be satisfied if it is possible to create links to new tables, correponding to new sub-clases, without re-compiling the existing programs. We shall return to this point below.

ORM4 This technique is a variation of ORM3 in that a *discriminator field* is used to record the name of the sub-class (or table) to which the object belongs. The use of discriminators is well-known from the representation of variant records. A composite primary key field is defined allowing for the support of multiple inheritance. As can be seen in Figure 5 an additional discriminator field is incorporated in each class to relational table representation. The advantage of this approach over ORM3 is that, for the cost of an additional column per superclass one can check the status of an object (what kind of employee an entry is) from within the table corresponding to the super-class. As in ORM3 before, (P1) and (P3) are satisfied and (P4) is more easily satisfied, since it is only necessary to visit another table if the object is known to be in a

Student	Employee	Casual
entity_id	entity_id	entity_id
name	name	name
address	address	address
hours	salary	salary
		level

Figure 3: ORM2: One table per concrete class

Entity	Student
entity_id	entity_id
name	course
address	

Employee	Casual
entity_id	entity_id
salary	hours

Figure 4: ORM3: One table per class

sub-class. That is, the creation of sub-classes that are not used imposes almost no overhead on existing programs.

Summarising, it appears that ORM4 has the greatest promise, provided one is able to add method specialisations to existing compiled code (P2). More generally, there is the issue of how to convert the fields and methods of a class definition into relations and queries.

3 The Pattern Calculus

The pattern calculus (Jay 2004*c*) supports pattern-matching functions in which different cases may have different type specialisations of a common default as well as supporting specialisation through sub-typing. This section will identify some fo the key features that are relevant to this paper.

The syntax of the *patterns* (meta-variable p) and *raw terms* (meta-variable t) of the pattern calculus is given by

$$\begin{array}{l} p ::= x \mid c \mid p\ p \\ t ::= x \mid c \mid t\ t \mid \mathsf{at}\ p\ \mathsf{use}\ t\ \mathsf{else}\ t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \end{array}$$

The *variables* are represented by the meta-variable x. The *constructors* (meta-variable c) are constants of the language which do not appear at the head of any evaluation rule. Other constants may be added if desired but their evaluation rules will not be considered explicitly in the formal development. The *application* $s\ t$ applies the function s to its argument t. The novel term form is the *extension* at p use s else t where p is the *pattern*, s is the *specialisation* and t is the *default.* The let-term let $x = s$ in t binds x to s in t. The declaration is recursive, in that free occurrences of x in s are bound to s itself.

Extensions combine abstraction over bound variables with a branching construction. For example, the λ-abstraction $\lambda x.s$ is short-hand for the extension

$$\mathsf{at}\ x\ \mathsf{use}\ s\ \mathsf{else}\ \mathsf{err}$$

where err is some form of error term, e.g. a non-terminating expression (such as let $x = x$ in x) or an exception.

The *substitution* $s\{u/x\}$ of a term u for a variable x in term s is defined in the usual way, as are bound variables and their α-conversion. The *terms* are defined to be equivalence classes of raw terms under α-conversion.

A *constructed term* is a term whose head is a constructor, i.e. a term which is either a constructor or of the form $t_1\ t_2$ in which t_1 is constructed. Let c be a constructor. A term u *cannot become* c if it is either a constructed term other than c or an extension. A term u *cannot become applicative* if it is either a constructor or an extension.

Entity	Student
entity_id	Student_id
name	course
address	discriminator
discriminator	

Employee	Casual
Employee_id	Casual_id
salary	hours
discriminator	discriminator

Figure 5: ORM4: One table per class with a discriminator field

$$\begin{array}{l}
(\mathsf{at}\ x\ \mathsf{use}\ s\ \mathsf{else}\ t)\ t_1 > s\{t_1/x\} \\
(\mathsf{at}\ c\ \mathsf{use}\ s\ \mathsf{else}\ t)\ c > s \\
(\mathsf{at}\ c\ \mathsf{use}\ s\ \mathsf{else}\ t)\ t_1 > t\ t_1 \\
\quad \text{if } t_1 \text{ cannot become } c \\
(\mathsf{at}\ p_1\ p_2\ \mathsf{use}\ s\ \mathsf{else}\ t)\ (t_1\ t_2) > \\
\quad (\mathsf{at}\ p_1 \\
\quad\quad \mathsf{use}\ \mathsf{at}\ p_2\ \mathsf{use}\ s\ \mathsf{else}\ \lambda y.t\ (p_1\ y) \\
\quad\quad \mathsf{else}\ \lambda x, y.t\ (x\ y) \\
\quad)\ t_1\ t_2 \quad \text{if } t_1 \text{ is a constructed term} \\
(\mathsf{at}\ p_1\ p_2\ \mathsf{use}\ s\ \mathsf{else}\ t)\ t_1 > t\ t_1 \\
\quad \text{if } t_1 \text{ cannot become applicative} \\
\mathsf{let}\ x = s\ \mathsf{in}\ t > t\{s/x\}
\end{array}$$

Figure 6: Reduction rules for the pattern calculus

The *basic reduction rules* of the constructor calculus are given by the relation $>$ in Figure 6. Let us consider the cases. Suppose that the pattern is a variable x. Specialisation is achieved by β-reduction, with the argument u being substituted for x in the specialisation. Suppose that the pattern is some constructor c and the argument is a constructed term u. If u is c then the specialisation is returned else the default is applied to u. Suppose that the pattern is an application $p_1\ p_2$ and the argument is a constructed term u. If u is an application $u_1\ u_2$ then specialisation tries to match p_1 with u_1 and p_2 with u_2; if either of these matches fails then evaluation reverts to applying the default to a reconstructed version of $u_1\ u_2$ (not $u_1\ u_2$ itself since this may require re-evaluation). If u is a constructor then the default is applied to it. Reduction of a let-term replaces the bound variable by its recursive definition.

4 The Discriminating Object Model

The approach adopted in the pattern calculus is to separate a class definition into a datatype declaration and some associated functions. The datatype definitions that correspond to the classes defined in Figure 1 are given in Figure 7. In turn, the datatype declarations for extensible classes employ a type variable to represent any additional fields that may be required. In particular, this type variable "is" the type of the discriminator field in the corresponding table. As can be seen by the use of the type variable X, Y, Z, and U in Figure 7 for any additional fields for the datatypes Entity, Student, Employee and Casual respectively.

```
datatype Entity X =
    Entity of X * int * string * string;;

datatype Student_Data Y =
    Student of Y * string;;

datatype Employee_Data Z =
    Employee of Z * float;;

datatype Casual_Data U =
    Casual of U * float;;
```

Figure 7: Class definitions as datatypes

In using the pattern calculus combined with ORM4 we can define one relational table per class. Therefore the creation of a new subclass does not require restructuring of objects. Method specialization, subtyping and type variable instantiation are all supported by the type safety of the Pattern Calculus. These coupled with ORM4 ensures that the implementation of an inherited method is unchanged by the creation of its subclass.

Let us now look at how our running example can be represented in this approach.

```
>-|> class Entity [a] {
       entity_id :  int;
       name :  string;
       address :  string; }

rest_Entity : Entity[a] → a
entity_id : Entity[a] → int
name : Entity[a] → string
address : Entity[a] → string
```

Figure 8: Entity class in bondi

Given in Figure 8 is the class definition of Entity as defined in **bondi**, along with the corresponding **bondi** session output. Programs typed by the user follow the prompt >-|>. System responses are given in italic. The representation of Entity and its associated fields are displayed along with the additional rest_Entity field. In the simplest case of an entity, the rest field is instantiated to be the unit type unit whose unique value can be called Null. In the database, the simplest case of an entity will correspond to a single row of data in the entity table where the discriminator field will be populated with Null.

In Figure 9 we see defined the Student class and its representation in **bondi**. A Student Entity has type Entity (Student_Data Y) or just Entity (Student_Data Unit) if the student has no additional fields. In the database, a Student Entity will correspond to an entry in both the entity and student tables. The discriminator field in the entity table will be populated with the string Student while in the student table the discriminator field in will be Null. The entity_id field is defined as both the primary and foreign key in the entity table as it's also the primary key field in the student table, that is the field student_id.

```
>-|> class Student [a] extends Entity {
       course :  string; }

rest_Student : Student[a] → a
course : Student[a] → string
```

Figure 9: Student class in bondi

```
>-|> classEmployee [a] extends Entity {
       salary :  float;
       salaryIncrease() =
          {this.salary * 1.10} }

rest_Employee : Employee[a] → a
salary : Employee[a] → float
salaryIncrease :
       Employee[a] * unit → float
```

Figure 10: Employee class in bondi

In Figure 10 is defined the Employee class and its field representations in **bondi**. An Employee Entity has type Entity (Employee_Data Y) or just Entity (Employee_Data Unit) if the employee has no additional fields. In the database, an Employee Entity will correspond to an entry in both the entity and employee tables. Similarly to how the discriminator field is used to represent student entity entries the same occurs for employee entity entries. That is the discriminator field in the employee table will be populated with the string Employee while in the employee table the discriminator field in will be Null, for ordinary employees. Note that the use of the discriminator field combined with the entity_id enables the support of multiple inheritance. A situation may arise where a student is also an employee. In this case they will have an entry in the entity, student and employee tables.

```
>-|> class Casual [a] extends Employee {
       hours :  float;
       salaryIncrease() =
           { this.salary * 1.06} }

rest_Casual : Casual[a] → a
hours : Casual[a] → float
salaryIncrease : Casual[a] * unit → float
```

Figure 11: Casual class in bondi

Figure 11 defines the Casual class and its field representations in **bondi**. The type of a casual employee is some Entity (Employee_Data (Casual_Data U)). In the database, an ordinary casual employee will correspond to an entry in each of the entity, employee and casual tables. The discriminator field in the entity table will be given as employee, in the employee table it will be given as casual and in the casual table will be given as Null. The same identity value will be used across all three tables as the entity_id, employee_id and casual_id respectively. Note that the type of a student or of a casual employee is always of the form Entity T for some type T, so that functions which act on arbitrary entities can always have the argument type Entity X. Again, each of these types has a single constructor. For example, that for entities is

Entity X : X* int *string * string → Entity X

Hence, a single pattern of the form Entity λx where λx is a binding variable will be able to match an arbitrary entity.

The notion of extensions in the pattern calculus allows us to specialise methods in the subclass without the need to recompile methods from the super class. This can best be exemplified through the salaryIncrease method defined in the running example. The salaryIncrease method defined in the Employee class is compiled to give a 10% increase in salary to all Employees as follows:

```
let (salaryIncrease :
    Entity (Employee_Data X) → float) =
    at Entity (Employee_Data (x,s),e,n,a)
    use s * 1.10
    else err;;
```

If the object is an employee entity that is of type Entity (Employee X) then a salary increase is calculated otherwise it will error.

This method defined in the superclass will translate into a stored procedure of something similar to the following code:

```
create procedure sp_salaryIncrease
begin transaction
       update Employee
       set salary = salary * 1.10
commit
```

Notice that since this method was defined for all employees the above sql code does not contain a where clause.

The specialisation of the salaryIncrease method in the Casual Employee class introduces a new case to the existing (compiled) method. Instead of recompiling the method for salaryIncrease, the pattern calculus defines a new method as follows:

```
let (newSalaryIncrease :
    Entity (Employee_Data X) → float) =
    at Entity(Employee_Data(Casual h y,s),
            e,n,a)
    use s * 1.06
    else salaryIncrease;;

let salaryIncrease = newSalaryIncrease;;
```

The new method encompasses the new case for Casual Employees as well as the existing code for ordinary Employees, without having to recompile the existing method.

We envisage that this method will translate into sql code such as the following:

```
create procedure sp_newSalaryIncrease
begin transaction
       if discriminator = "Casual" then
           update Employee
           set salary = salary * 1.06
           where discriminator = "Casual"
       else if discriminator = NULL then
           sp_salaryIncrease
endif
commit

sp_rename newSalaryIncrease salaryIncrease;;
```

Similarly to how we are able to translate the notion of the type variables in the programming language with discriminator fields in the database, we can see that notion of separate compilation can be applied to queries similarly to how they apply to methods. This is made possible due to the combination of a number of features.

The linking of the data model of the pattern calculus to that in the database through ORM4 allows for the support of the four desirable properties. Efficient data storage, one relation per class and modular compilation all are attained. The final property (P4) of compiled methods taking advantage of queries appears promising from the manual translation of the salaryIncrease method.

5 Conclusion and Future Work

The discriminating object model is able to represent objects and classes in both programming and relational databases. Each class is represented by a single table whose fields are those of the class plus a discriminator field to indicate which sub-class, if any, the object belongs to. The addition of new sub-classes does not change the relation itself, but merely creates new options for the discriminator. Hence, the relationship between the class and the database is stable, and so compilation can be modular. Also, the addition of the discriminator field has little or no impact on efficiency. This approach has been implemented in **bondi**.

The benefits (P1–P3) derived from the discriminating object model are not limited to a particular account of object-orientation and should prove useful for a variety of languages.

Future work will explore techniques for compiling simple methods into queries. This is particularly suited to development in the pattern calculus, as it supports an incremental approach to defining methods. Further, their execution can be driven by the discriminators without consulting any class hierachy in the programming language. If successful, this would combine the expressive power of the object-oriented programming style with the efficiency of query languages.

References

Agrawal, R., Dar, S. & Gehani, N. H. (1993), The o++ database programming language: Implementation and experience, Technical Report 61-70, ATT Bell Laboratories.

Ambler, S. W. (2003), *Agile Database Techniques: Effective Strategies for the Agile Software Developer*, Wiley, chapter 14.

Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D. & Zdonik, S. (1994), 'The object-oriented database system manifesto'. http://www-2cs.cmu.edu/People/clamen/OODBMS/ Manifestor/htManifesto/node31.htm.

Bierman, G., Meijer, E. & Schulte, W. (2005), 'The essence of data acces in cω. the power is in the dot!', http://research.microsoft.com/ emeijer/Papers/popl.pdf. (Accepted for publication at ECOOP 2005).

Bloom, T. & Zdonik, S. T. (1987), 'Issues in the design of object-oriented database programming languages', *OOPSLA '87 Proceedings* pp. 441-451.

Breazu-Tannen, V., Buneman, P. & Ohori, A. (1991), Data structures and data types for object-oriented databases, *in* 'IEEE Data Engineering bulletin, Special Issue on Theoretical Foundations of Object-Oriented Database Systems', Vol. 14, IEEE, pp. 23–27.

Fussell, M. L. (1997), Foundations of object relational mapping, Technical report, Chimu publications.

Jay, C. B. (2004*a*), 'bondi', www-staff.it.uts.edu.au/ cbj/bondi.

Jay, C. B. (2004*b*), Methods as pattern-matching functions, *in* 'Foundations of Object-Oriented Languages, 2004: informal proceedings', p. 16 pp. http://www.doc.ic.ac.uk/ scd/FOOL11/ patterns.pdf.

Jay, C. B. (2004*c*), 'The pattern calculus', *ACM Transactions on Programming Languages and Systems (TOPLAS)* **26**(6), 911–937.

Jay, C. B. (2004*d*), 'Unifiable subtyping', www-staff.it.uts.edu.au/ cbj/Publications/ unifiable_subtyping.pdf.

Keller, W. (1998), Object/relational access layers - a roadmap, missing links and more patters, *in* 'EuroPLoP'.

Leontiev, Y., Ozsu, M. & Szafron, D. (2002), 'On type systems for object-oriented database programming languages', *ACM Computing Surveys* **34**(4), 409–449.

Remy, D. (1989), Typechecking records and variants in a natural extension of ML, *in* L. Cardelli, ed., 'Proceedings of POPL'98, ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages', ACM.

Richardson, J. E., Carey, M. J. & Schuh, D. T. (1989), The design of the e programming language, Technical report, University of Wisconsin.

Tracing Secure Information Flow Through Mode Changes

Colin Fidge[1] **Tim McComb**[2]

[1]School of Software Engineering and Data Communications, Queensland University of Technology
[2]School of Information Technology and Electrical Engineering, The University of Queensland

Abstract

Communications devices intended for use in security-critical applications must be rigorously evaluated to ensure they preserve data confidentiality. This process includes tracing the flow of classified information through the device's circuitry. Previous work has shown how this can be done using graph analysis techniques for each of the device's distinct operating modes. However, such analyses overlook potential information flow between modes, via components that store information in one mode and release it in another. Here we show how graph-based analyses can be extended to allow for information flow through sequences of consecutive modes.

1 Introduction

Electronic communications devices safeguard classified information in government and military computer networks. In particular, 'domain separation' devices allow the flow of information between high and low-security networks to be controlled. Examples of such devices include: data diodes, which allow one-way information flow only; multi-computer switches, which allow peripheral devices to be shared between different domains; context filters, which constrict information flow; and cryptographic devices, which allow transmission of classified information over insecure networks.

Before such a device can be deployed, however, it must be carefully evaluated to ensure that it maintains data confidentiality. Such evaluations involve a detailed analysis of the device's design and construction (Bishop 2003, Ch. 21), as prescribed by international security standards (Herrmann 2003).

Previous work has shown how information flow can be traced through the circuitry of domain-separation devices to see if there are unintended data pathways from a high-security domain to a low-security one (Rae & Fidge 2005*a*, Rae & Fidge 2005*b*, McComb & Wildman 2005). Because a device will typically send data to different destinations in different modes, such analyses must take each of the device's different operating modes into account. These include 'failure' modes in which a faulty component alters the flow of information.

We wish to thank the anonymous reviewers for their helpful comments, especially a suggestion which inspired the example in Section 8. This research was funded by the Defence Signals Directorate and the Australian Research Council via Linkage-Projects Grant LP0347620, *Formally-Based Security Evaluation Procedures.*

However, previous analyses fail to identify unintended information-flow pathways created by *changes* in operating mode. A potential security risk may be caused by components within a device that can store information in one mode and release it in another. It is then possible for classified data en route to a high-security destination to be accidentally diverted to a low-security domain when the device's mode is changed.

Here we use a series of worked examples to show how a simple extension to existing graph-theoretic analysis techniques allows us to identify potential security leaks caused by mode changes. This is done by allowing information-flow pathways to be defined not only as those in which data is transmitted in a particular operating mode, but also as those in which data is communicated *between* different operating modes, providing the information flows intersect at components that may store and forward data.

2 Previous Work

International standards such as the *Common Criteria for Information Technology Security Evaluation* define the need to undertake detailed analyses of communications devices intended for use in high-security computer networks (Common Criteria 1999). Inevitably, however, general standards like these offer broad guidelines only, so much work is required to put them into practice (Herrmann 2003).

Our own research has been dedicated to automatable evaluation techniques for domain-separation devices. In previous work we have shown how to analyse electronic circuitry to identify information flow paths (Rae & Fidge 2005*a*, Rae & Fidge 2005*b*), and have implemented this theory in a practical tool (McComb & Wildman 2005). However, this work was limited to exploring information flow occurring within a single operating mode of the device. Behaviours that span modes were not considered.

In the context of safety-critical, rather than security-critical, systems the 'mode logic' of embedded controllers has been well explored elsewhere (Miller 1998, Paynter 1996). Although similar in motivation to our research, this work is not concerned with information flow, but rather with the need to model the way the system may change operating modes, in order to prove that no undesirable sequences of mode changes are possible.

3 Motivation

In previous research we explained how the schematic circuit diagram of a communications device could be treated as an information flow graph for the purposes of security evaluation (Rae & Fidge 2005*b*). Discrete components such as logic gates and microprocessor chips are treated as nodes, and the wiring connecting them as arcs. The way each type of component connects its inputs to its outputs is defined for each of the device's operating modes. The device's circuit diagram is then modelled as an adjacency matrix of the following form.

	outputs
inputs	modes

Each row represents a connection acting as an input and each column a connection acting as an output. Based on the behaviour of the various components, the cells in the matrix are populated with sets of operating modes, possibly empty, defining the particular modes in which the corresponding input and output are connected.

Connectivity across the whole circuit is then defined as the transitive closure of the adjacency matrix. The security evaluator can then easily see whether there is any information flow from those inputs to the device which come from a high-security domain and those outputs which lead to a low-security domain.

Importantly, the transitive closure is calculated under the assumption that two adjacent arcs are connected only if they are both 'active' in the same mode. Since a circuit diagram is usually fully-connected, this assumption limits the information flow analysis to realistic cases.

Although adequate for components such as logic gates, that can be modelled as if they 'instantaneously' transfer information from their inputs to their outputs, this assumption is not always satisfactory. If it possible for a particular component to store information when the device is one mode, and subsequently release the information when the device is in another mode, then classified data may flow through the device in a way that will not be identified by a mode-specific analysis. Therefore, our goal below is to extend the approach to allow for the possibility that some components, such as memory chips and flip flops, can transfer information from one operating mode to another.

4 Tracing Information Flow Through Mode Changes

To do this we must extend the notion that a component connects its inputs to its outputs in a particular mode to allow for components that receive information in one mode and release it in a subsequent mode. Therefore, rather than reasoning about particular modes of the device, we must consider *sequences* of consecutive modes.

Let $M, N, \ldots$ represent distinct operating modes of a device. In practice these may be 'normal' behaviours, fault modes or both. We firstly introduce mode sequences within angled brackets, e.g., $\langle M, N, O, P\rangle$, to model the notion that the device has switched between the particular sequence of modes, in the order shown.

To model the mode-specific way a component connects its inputs to its outputs we also need a special operator for joining mode sequences end-to-end. This is denoted here by '$X \oplus Y$', which joins (non-empty) mode sequences X and Y provided that the last mode in sequence X is the same as the first mode in sequence Y.

$$\begin{aligned} & \langle M, \ldots, O, P\rangle \oplus \langle P, Q, \ldots, S\rangle \\ = \; & \langle M, \ldots, O, P, Q, \ldots, S\rangle \end{aligned} \tag{1}$$

Sequences whose last/first modes don't match cannot be joined. For example, $\langle M, \ldots, P\rangle \oplus \langle Q, \ldots, S\rangle$ does not produce a valid mode sequence.

To model components that may exchange information in *any* mode, we also introduce mode 'wildcards', denoted '$\star$' that match any mode.

$$\begin{aligned} & \langle M, \ldots, O, P\rangle \oplus \langle \star, Q, \ldots, S\rangle \\ = \; & \langle M, \ldots, O, P, Q, \ldots, S\rangle \end{aligned} \tag{2}$$

Similarly for the symmetric case with the wildcard at the end of the first sequence. Notice that, given this rule, we can use the sequence $\langle \star, \star\rangle$ as a 'connector' to join any two mode sequences.

Since it is unhelpful to say that a device switches from a given mode P to the same mode P, we also assume that duplicated, non-wildcard modes are always compressed.

$$\begin{aligned} & \langle M, \ldots, O, P, P, Q, \ldots, S\rangle \\ = \; & \langle M, \ldots, O, P, Q, \ldots, S\rangle \end{aligned} \tag{3}$$

We then follow our previous approach (Rae & Fidge 2005*b*) of populating the adjacency matrix with the mode-specific connections implied by the individual components in the device and performing matrix multiplications to calculate the transitive closure and achieve an end-to-end connectivity analysis. Now, however, the cells of the adjacency matrix contain sets of mode sequences, rather than just sets of modes.

Let Z be an $N \times N$ adjacency matrix. Let $Z_{(i,j)}$ be the value in the cell in Z's ith row and jth column. Then we define the square Z^2 as a matrix in which the cell in the ith row and jth column is calculated as follows.

$$Z^2_{(i,j)} = \bigcup_{1 \le k \le N} \{s \oplus t \mid s \in Z_{(i,k)} \wedge t \in Z_{(k,j)}\} \tag{4}$$

In other words, if information can flow from input i to output k via mode sequence s, and information can flow from input k to output j via mode sequence t, then information can flow from input i to output j via mode sequence $s \oplus t$ (provided that sequences s and t can be joined using Rules 1 to 3 above). The transitive closure of a graph can be found by squaring its adjacency matrix until it stops changing.

To illustrate the technique and its properties, the following sections present four distinct examples in which 'inter-mode' information flow may occur.

5 Example: A Device With a Modal, Buffered Component

Here we consider an example where a component with memory transmits information in different directions in different operating modes. An abstract view of the device of interest is shown in Figure 1. It comprises four components, A, B, C and D, which act as serial interface converters, and a switching component X, which directs the flow of information. Such a device may, for instance, form part of a multi-computer switch, used to allow peripheral devices to be shared between different security domains, while still maintaining separation of data.

We assume that component X has two different operating modes, M and N, selected by a physical switch on the device (not shown in the figure). In mode M information flows along connections e, f, g and h, via components B, X and A. In mode N information flows along connections i, j, k and l, via components D, X and C. Connections e and h interface to a high-security domain, while connections i

Table 1: Adjacency matrix for the switching device using global modes

	e	f	g	h	i	j	k	l
e		$\{M\}$						
f			$\{M\}$					
g				$\{M\}$				
h								
i						$\{N\}$		
j							$\{N\}$	
k								$\{N\}$
l								

Table 2: Connectivity matrix for the switching device using global modes

	e	f	g	h	i	j	k	l
e		$\{M\}$	$\{M\}$	$\{M\}$				
f			$\{M\}$	$\{M\}$				
g				$\{M\}$				
h								
i						$\{N\}$	$\{N\}$	$\{N\}$
j							$\{N\}$	$\{N\}$
k								$\{N\}$
l								

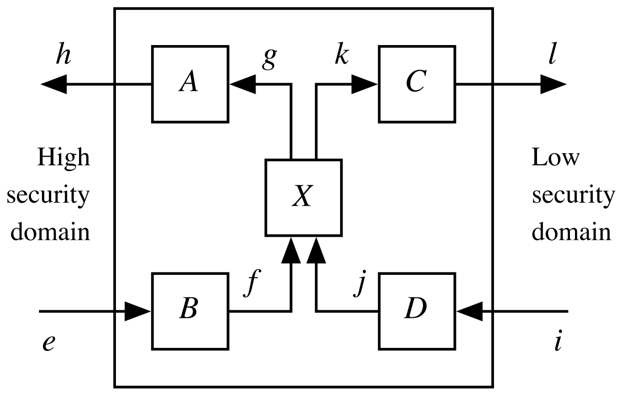


Figure 1: A buffered switching device

and l interface to a low-security domain. We also assume that component X contains a data buffer, to smooth the flow of data packets through the device.

To evaluate such a device's security properties we must determine whether there are any information-flow pathways from the high-security domain to the low-security one in any operating mode. We first perform the analysis using the previous mode-specific approach (Rae & Fidge 2005*b*) to illustrate its weakness.

Table 1 shows the initial adjacency matrix as populated by the security evaluator, showing connectivity through the individual components in different modes. For instance, it says that information flows from e to f, from f to g and from g to h in mode M, as per the above description of the device's behaviour. Similarly for mode N. All unoccupied cells are assumed to contain the empty set of modes. We omit the cells on the diagonal since it is unhelpful to note that an arc is connected to itself.

To analyse the device's overall connectivity, we then perform matrix multiplications until a fixed point is reached, populating each cell (i, j) with the intersection of the mode sets from cells (i, k) and (k, j) (Rae & Fidge 2005*b*). This is consistent with the notion that components must be in the same mode to interact. The resulting connectivity matrix is shown in Table 2.

This analysis correctly shows that information may flow from e to h in mode M and from i to l in mode N. Moreover, the two clusters of occupied cells in Table 2 clearly suggests that the device successfully achieves domain-separation in its two modes.

Unfortunately, this analysis completely overlooks the possibility that component X retains information between modes. To solve this we redo the analysis using mode sequences as defined in Section 4 above.

Table 3 shows our initial population of the adjacency matrix using mode sequences. As before it includes the explicit information flow in operating modes M and N. However, this time the security evaluator has chosen to use 'mode wildcard' sequences to indicate that buffered component X *may* forward information received on input j in one mode to output g in another mode. Similarly for input f and output k. (We have added only those additional connections created by component X's ability to store-and-forward information in different modes. Adding 'wildcards' to the (f, g) and (j, k) cells would also be reasonable but does not change the outcome in this case.)

Table 4 then shows the matrix after multiplying twice (at which point it stops changing) using Rule 4 above. This time the (i, h) cell tells us that information may flow between these two arcs when component X's mode is switched from N to M. This is an unexpected but harmless behaviour of the device.

Of much more concern is that the (e, l) cell shows potential information flow when the mode changes from M to N. This is especially disturbing since this is a previously unsuspected flow from the high-security domain to the low-security one.

In this way the security evaluator is alerted to a potentially dangerous flow of information and the need to study this pathway in more depth. For instance, if it can be proven that component X's buffer is cleared

Table 3: Adjacency matrix for the switching device using mode sequences

	e	f	g	h	i	j	k	l
e		$\{\langle M\rangle\}$						
f			$\{\langle M\rangle\}$				$\{\langle\star,\star\rangle\}$	
g				$\{\langle M\rangle\}$				
h								
i						$\{\langle N\rangle\}$		
j			$\{\langle\star,\star\rangle\}$				$\{\langle N\rangle\}$	
k								$\{\langle N\rangle\}$
l								

Table 4: Connectivity matrix for the switching device using mode sequences

	e	f	g	h	i	j	k	l
e		$\{\langle M\rangle\}$	$\{\langle M\rangle\}$	$\{\langle M\rangle\}$			$\{\langle M,\star\rangle\}$	$\{\langle M,N]\rangle\}$
f			$\{\langle M\rangle\}$	$\{\langle M\rangle\}$			$\{\langle\star,\star\rangle\}$	$\{\langle\star,N\rangle\}$
g				$\{\langle M\rangle\}$				
h								
i			$\{\langle N,\star\rangle\}$	$\{\langle N,M\rangle\}$		$\{\langle N\rangle\}$	$\{\langle N\rangle\}$	$\{\langle N\rangle\}$
j			$\{\langle\star,\star\rangle\}$	$\{\langle\star,M\rangle\}$			$\{\langle N\rangle\}$	$\{\langle N\rangle\}$
k								$\{\langle N\rangle\}$
l								

every time the device changes from mode M to N then the device may still be considered secure. If not, however, it may need to be rejected.

The transitive closure also produces 'wildcard' mode sequences in the (i,g), (j,h), (e,k) and (f,l) cells, denoting potential information flow along these paths in different modes, but since none of these paths both begin and end in the device's environment they are of little security interest. Generally, we are interested only in pathways leading from a high-security domain, or some other source of classified information, to a low-security domain.

6 Example: A Device with a Non-Moded Buffered Component

In this section we consider an example where the component that retains information between modes is not itself affected by mode changes, but another component that uses it is.

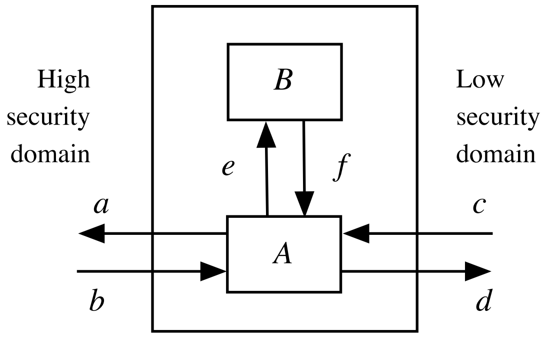


Figure 2: A cryptographic device

Figure 2 shows a (highly abstract) view of a cryptographic device. It connects a high-security computer to a low-security network. The device has two modes, encryption E and decryption D. In encryption mode E plaintext messages are read from the computer via input b, encrypted by processor A, using memory chip B to temporarily store each block of the resulting ciphertext, and these are then sent to the network via output d. In decryption mode D ciphertext messages are read from the network via input c, decrypted by processor A, using memory chip B to temporarily store each block of the resulting plaintext, and these are then sent to the computer via output a. In practice, such devices are used in pairs, allowing computers in two different high-security domains to communicate over a low-security network. Each device must thus perform both encryption and decryption functions to support bidirectional communication. The device's overall mode, encryption or decryption, is controlled by signals (not shown) sent to processor A from either the local or remote high-security domain.

Once again, we first attempt to evaluate the device using the old approach. Table 5 shows the security evaluator's population of the adjacency matrix, based on our understanding of the way components A and B work. To model the fact that information can be written to and subsequently read from memory chip B in either mode, we put both modes E and D into the (e,f) cell. (This seems unnatural, because a memory chip doesn't normally have 'modes', but leaving the cell empty would imply there is no information flow through this component. Indeed, this dilemma highlights how poorly suited the previous approach is to handling devices that store-and-forward information.)

The transitive closure after performing matrix multiplications is shown in Table 6. It correctly identifies information flow from c to a in decryption mode D and from b to d in encryption mode E, both of which are expected for this device.

Again, however, this approach fails to recognise the possibility that information stored in the memory chip in one mode may accidently be released in another mode. Therefore, we redo the analysis using mode sequences. To model the memory component's ability to connect information flow in any

Table 5: Adjacency matrix for the cryptographic device using global modes

	a	b	c	d	e	f
a						
b					$\{E\}$	
c					$\{D\}$	
d						
e						$\{E, D\}$
f	$\{D\}$			$\{E\}$		

Table 6: Connectivity matrix for the cryptographic device using global modes

	a	b	c	d	e	f
a						
b				$\{E\}$	$\{E\}$	$\{E\}$
c	$\{D\}$				$\{D\}$	$\{D\}$
d						
e	$\{D\}$			$\{E\}$		$\{E, D\}$
f	$\{D\}$			$\{E\}$		

pair of modes—because the memory chip itself is not influenced by changes to the device's encryption/decryption mode—we put a wildcard entry into the (e, f) cell in Table 7.

The resulting transitive closure after matrix multiplication is shown in Table 8. This time the (b, a) cell tells us that information put into memory in encryption mode E could potentially be sent back to the high-security domain in decryption mode D. Although this is not an intended behaviour of the device, it would be harmless from a security perspective.

More seriously, though, the (c, d) cell alerts us to the danger that

1. processor A, in decryption mode, reads a ciphertext message from input c,
2. it decrypts the ciphertext and stores the resulting blocks of plaintext data in memory chip B via arc e,
3. processor A's mode is switched to encryption,
4. processor A reads the contents of memory component B, via arc f, and sends the blocks of decrypted plaintext to the low-security domain via output d.

This series of events would allow the device to decrypt a ciphertext message received from the low-security domain and send the resulting plaintext back to the low-security domain!

Having been alerted to this risk, the security evaluator would be obliged to look for mitigations against it. This would most likely require a careful analysis of the software on processor A to ensure that it accesses arrays in memory in a safe way or, better still, that it completely clears the contents of memory chip B when the mode changes from encryption to decryption.

7 Example: Combining Buffered and Modal Modules

The next example demonstrates that the approach has good compositional behaviour. In this case we show that combining a buffered component with an otherwise 'secure' modal circuit makes the overall device insecure in the presence of mode changes.

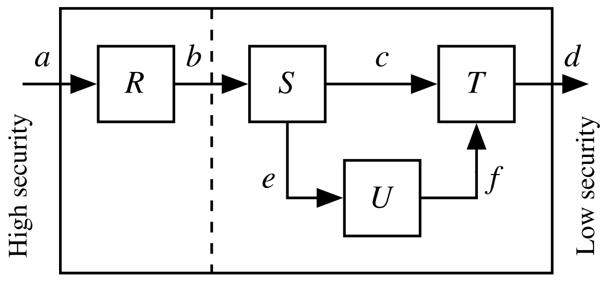


Figure 3: A context filtering device

Figure 3 shows a (highly abstract) design for a context filter. Such devices are used to restrict the flow of information from a high-security domain to a low-security one. In this case we assume the device has two operating modes, filter F and bypass B. When the device is in filter mode it is intended to accept information from the high-security domain and forward only data packets that satisfy some predefined security criterion to the low-security domain. When the device is in bypass mode information may flow from the high-security to the low-security domain without restriction. (Bypass modes are typically needed in such devices to allow binary control data through while setting up communication links.)

The device's design consists of two main modules. On the right is a modal (and essentially memoryless) module in which component S directs the flow of information depending on whether the device is in bypass or filter mode. Component T is a simple merge component (probably just an 'or' gate in practice). Component U performs the filtering function, which normally involves excising messages being sent from the high-security domain to the low-security one if they are not in some dictionary of allowed messages. On the left of Figure 3 is a separate module containing an unmoded input buffer R which is used to smooth the flow of traffic into the filter.

The interesting aspect of this device is that, in isolation, both the left and right-hand modules have well-understood, 'secure' behaviours, but their composition in the presence of mode changes does not.

Table 7: Adjacency matrix for the cryptographic device using mode sequences

	a	b	c	d	e	f
a						
b					$\{\langle E\rangle\}$	
c					$\{\langle D\rangle\}$	
d						
e						$\{\langle\star,\star\rangle\}$
f	$\{\langle D\rangle\}$			$\{\langle E\rangle\}$		

Table 8: Connectivity matrix for the cryptographic device using mode sequences

	a	b	c	d	e	f
a						
b	$\{\langle E,D\rangle\}$			$\{\langle E\rangle\}$	$\{\langle E\rangle\}$	$\{\langle E,\star\rangle\}$
c	$\{\langle D\rangle\}$			$\{\langle D,E\rangle\}$	$\{\langle D\rangle\}$	$\{\langle D,\star\rangle\}$
d						
e	$\{\langle\star,D\rangle\}$			$\{\langle\star,E\rangle\}$		$\{\langle\star,\star\rangle\}$
f	$\{\langle D\rangle\}$			$\{\langle E\rangle\}$		

Table 9 shows our initial population of the adjacency matrix for this device using mode sequences. In particular, the (a,b) cell shows that component R can receive information in either mode B or F and release it in any mode.

(Putting $\langle\star,\star\rangle$ in this cell would have exactly the same meaning, since the only two modes are B and F, but explicitly enumerating the modes for cells that are on the outermost boundary of a diagram produces a more readable result. Another modelling decision in this table is that the evaluator has chosen to separately specify the two distinct modes in which information is *intended* to flow through component T, either from c to d in mode B or from f to d in mode F. Although reasonable, we could equally well take the view that this simple 'or' gate-like component is essentially unmoded and put both modes B and F in the (c,d) and (f,d) cells. This has no significant impact on the overall example, however.)

Table 10 then shows the calculated connectivity matrix. If we consider just the b to f rows and columns, we can see the overall connectivity for the right-hand module in Figure 3. In particular, the (b,d) cell tells us that this unbuffered module is 'secure' in the sense that information flows from end-to-end only in modes B or F, as we expect from such a device.

However, with the addition of buffered component R (i.e., the 'a' row and column in Table 10) we see that the device as a whole is *not* secure. Specifically, the $\langle F,B\rangle$ sequence in the (a,d) cell alerts us to the fact that information accumulated in component R while the device is in filter mode F can be subsequently released to the low-security domain when the device is switched into bypass mode B, and thus won't get filtered as the operator intended.

Once again, therefore, the security evaluator will be obliged to carefully consider the device's behaviour when it switches from filter to bypass mode. Although component S is the primary one concerned with the device's overall mode, we must ensure that all data in component R is cleared when the mode is changed. If this cannot be proven by more detailed analysis of the device's design it must be rejected as insecure.

8 Example: Clearing a Buffered Component

In the preceding examples we have assumed a worst-case scenario in which a buffered component may release any information acquired in previous modes. If, however, we know that the buffer is cleared when certain mode changes occur we can incorporate this information in the adjacency matrix and thus eliminate false positives from the connectivity analysis. Using sets of mode sequences in the cells makes this possible because we can selectively omit sequences from the sets when we know that this particular sequence of modes never allows information flow.

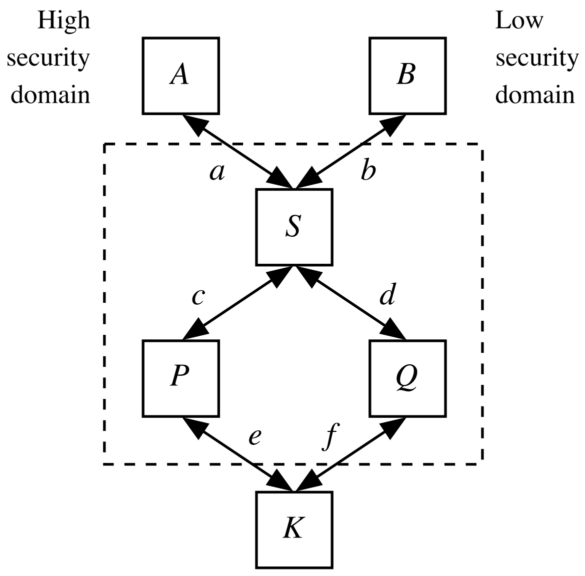


Figure 4: A keyboard switch

For example, consider the (highly abstract) design for a keyboard switch in Figure 4. Its purpose is to allow a single keyboard K to be shared between high-security computer A and low-security computer B. The switching device contains three components, a 'mode switch' S, and two microprocessors, P and Q, for interfacing between the keyboard and the high and low-security computers, respectively.

Table 9: Adjacency matrix for the filter device

	a	b	c	d	e	f
a		$\{\langle F,\star\rangle,\langle B,\star\rangle\}$				
b			$\{\langle B\rangle\}$		$\{\langle F\rangle\}$	
c				$\{\langle B\rangle\}$		
d						
e						$\{\langle F\rangle\}$
f				$\{\langle F\rangle\}$		

Table 10: Connectivity matrix for the filter device

	a	b	c	d	e	f
a		$\{\langle F,\star\rangle,\langle B,\star\rangle\}$	$\{\langle B\rangle,\langle F,B\rangle\}$	$\{\langle F\rangle,\langle B\rangle,$ $\langle F,B\rangle,\langle B,F\rangle\}$	$\{\langle F\rangle,\langle B,F\rangle\}$	$\{\langle F\rangle,\langle B,F\rangle\}$
b			$\{\langle B\rangle\}$	$\{\langle B\rangle,\langle F\rangle\}$	$\{\langle F\rangle\}$	$\{\langle F\rangle\}$
c				$\{\langle B\rangle\}$		
d						
e				$\{\langle F\rangle\}$		$\{\langle F\rangle\}$
f				$\{\langle F\rangle\}$		

The device has two modes, H and L, where the keyboard is connected to the high-security computer and the low-security computer, respectively. The mode of choice is controlled by the operator, using a physical switch connected to component S, which in turn sends appropriate control signals (not shown) to the two microprocessors.

In such a device, component S does not physically disconnect the keyboard from the 'disconnected' computer, to avoid problems caused by the computer detecting that its keyboard has been unplugged. Instead, the microprocessor linked to the disconnected computer simulates the behaviour of the keyboard so that both computers believe they have a dedicated keyboard attached at all times. For instance, when the device is in high-security mode H, microprocessor P acts as an interface between the keyboard and computer A, while microprocessor Q simultaneously simulates an idle keyboard for computer B.

Primary information flow through the keyboard switch consists of keystrokes forwarded from the keyboard to whichever computer is 'connected', depending on the operating mode. However, the computers also send signals to the keyboard, to drive LED and LCD displays, maintain the 'Caps-Lock' status, and to store information about programmable function keys. This is why all arcs in Figure 4 are shown as bidirectional. For instance, when the device is in mode H, not only do keystrokes travel from component K to component A, via components P and S, but some data also travels from the high-security computer to the keyboard in the reverse direction.

The presence of even a small memory buffer in the keyboard thus introduces the danger of a covert channel via which classified information can be sent from the high-security computer to the keyboard while the device is in mode H and subsequently forwarded from the keyboard to the low-security computer when the device is in mode L. Indeed, undertaking a connectivity analysis of this device, using 'wildcard connectors' to model information flow through the keyboard, would reveal that all the components in this device are potentially connected.

In this example, however, we wish to introduce the additional knowledge that a safeguard has been built into the device. Assume that the device has been engineered so that when it is switched from mode H to mode L, microprocessor P instructs the keyboard to clear all of its memory buffers.

We can incorporate this knowledge into our model as shown by the adjacency matrix in Table 11. Instead of putting '$\langle\star,\star\rangle$' connectors into the (e, f) and (f, e) cells, we have instead included only the possibility that information flows through the keyboard when the mode changes from L to H. Omitting the $\langle H, L\rangle$ sequence from these cells models our assumption that information flow is prevented when switching from mode H to mode L.

(Also note that component S is assumed to successfully keep the high and low-security data streams separate. There is no information flow between arcs a and d or between arcs b and c.)

The resulting transitive closure in Table 12 then shows that this device can be considered secure thanks to the assumption we have made about clearing the buffer. The (b, a) cell shows that information may flow from the low-security computer to the high-security one when the mode is switched from L to H, but this is harmless. Most importantly, the (a, b) cell is empty. Even though the (a, e) cell shows that information may flow from the high-security computer to the keyboard in mode H, and the (f, b) cell shows that information may flow from the keyboard to the low-security computer in mode L, our deliberate omission of a $\langle H, L\rangle$ link in the (e, f) cell successfully captured the notion that that the keyboard does not leak classified data in this design.

9 Conclusion

Evaluating information flow through communications devices intended for secure applications is both tedious and intellectually taxing. Our research is devising ways of automating various aspects of the problem. In this short paper we have described an improvement to previous analysis techniques which will help highlight potential security problems in devices which possess both different operating modes and the ability to temporarily store classified information between modes.

Although calculation of connectivity matrices is readily automatable, population of the adjacency ma-

Table 11: Adjacency matrix for the keyboard switch

	a	b	c	d	e	f
a			$\{\langle H\rangle\}$			
b				$\{\langle L\rangle\}$		
c	$\{\langle H\rangle\}$				$\{\langle H\rangle\}$	
d		$\{\langle L\rangle\}$				$\{\langle L\rangle\}$
e			$\{\langle H\rangle\}$			$\{\langle L,H\rangle\}$
f				$\{\langle L\rangle\}$	$\{\langle L,H\rangle\}$	

Table 12: Connectivity matrix for the keyboard switch

	a	b	c	d	e	f
a			$\{\langle H\rangle\}$		$\{\langle H\rangle\}$	
b	$\{\langle L,H\rangle\}$		$\{\langle L,H\rangle\}$	$\{\langle L\rangle\}$	$\{\langle L,H\rangle\}$	$\{\langle L\rangle\}$
c	$\{\langle H\rangle\}$				$\{\langle H\rangle\}$	
d	$\{\langle L,H\rangle\}$	$\{\langle L\rangle\}$	$\{\langle L,H\rangle\}$		$\{\langle L,H\rangle\}$	$\{\langle L\rangle\}$
e	$\{\langle H\rangle\}$		$\{\langle H\rangle\}$			$\{\langle L,H\rangle\}$
f	$\{\langle L,H\rangle\}$	$\{\langle L\rangle\}$	$\{\langle L,H\rangle\}$	$\{\langle L\rangle\}$	$\{\langle L,H\rangle\}$	

trix used as a starting point inevitably relies on a degree of skill from the security evaluator to model assumed device characteristics properly. Including too many modes in the cells will produce false positives, while omitting modes in which information is actually transmitted through a component runs the more serious risk of overlooking security problems. If in doubt about how to model a particular component, therefore, the evaluator would be best advised to put in too many modes, rather than too few.

All of the examples presented above were trivial block-diagram abstractions of the types of devices we normally consider. In practice, information security evaluations are based on circuitry schematics with dozens of distinct components. Such evaluations are practical only with tool support. We have already built a software tool capable of analysing circuit-level system descriptions (McComb & Wildman 2005) and now plan to extend it with the analysis capability described above.

In future work, this general area of research can be extended in two directions. Firstly, since we treat circuitry merely as a connectivity graph, the approach can also be applied to whole communications networks, not just single communications devices. Also, we plan to extend the type of analysis described above to embedded software running on microprocessors within communications devices, since embedded code has a major influence on the transfer of information between modes.

References

Bishop, M. (2003), *Computer Security: Art and Science*, Addison-Wesley.

The Common Criteria Project Sponsoring Organisations (1999), *Common Criteria for Information Technology Security Evaluation*, 2.1 edn. ISO/IEC Standard 15408.

Herrmann, D. S. (2003), *Using the Common Criteria for IT Security Evaluation*, Auerbach Publications.

McComb, T. & Wildman, L. P. (2005), SIFA: A tool for evaluation of high-grade security devices, *in* C. Boyd & J. Nieto, eds, 'Information Security and Privacy: Tenth Australasian Conference (ACISP 2005)', Vol. 3574 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 230–241.

Miller, S. P. (1998), Specifying the mode logic of a flight guidance system in CoRE and SCR, *in* M. Ardis, ed., 'Proceedings of FMSP'98: The Second Workshop on Formal Methods in Software Practice', ACM Press, pp. 44–53.

Paynter, S. (1996), Real-time mode-machines, *in* B. Jonsson & J. Parrow, eds, 'Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'96)', Vol. 1135 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 90–109.

Rae, A. J. & Fidge, C. J. (2005*a*), 'Identifying critical components during information security evaluations', *Journal of Research and Practice in Information Technology* **37**(4), 391–402.

Rae, A. J. & Fidge, C. J. (2005*b*), 'Information flow analysis for fail-secure devices', *The Computer Journal* **48**(1), 17–26.

SPiKE: Engineering Malware Analysis Tools using Unobtrusive Binary-Instrumentation

Amit Vasudevan **Ramesh Yerraballi**

Department of Computer Science and Engineering
University of Texas at Arlington,
Box 19015, 416 Yates St., 300 Nedderman Hall,
Arlington, TX 76019-0015, USA.
Email: {vasudeva, ramesh}@cse.uta.edu

Abstract

Malware — a generic term that encompasses viruses, trojans, spywares and other intrusive code — is widespread today. Malware analysis is a multi-step process providing insight into malware structure and functionality, facilitating the development of an antidote. *Behavior monitoring*, an important step in the analysis process, is used to observe malware interaction with respect to the system and is achieved by employing dynamic coarse-grained binary-instrumentation on the target system. However, current research involving dynamic binary-instrumentation, categorized into probe-based and just-in-time compilation (JIT), fail in the context of malware. Probe-based schemes are not transparent. Most if not all malware are sensitive to code modification incorporating methods to prevent their analysis and even instrument the system themselves for their functionality and stealthness. Current JIT schemes, though transparent, do not support multithreading, self-modifying and/or self-checking (SM-SC) code and are unable to capture code running in kernel-mode. Also, they are an overkill in terms of latency for coarse-grained instrumentation.

To address this problem, we have developed a new dynamic coarse-grained binary-instrumentation framework codenamed SPiKE, that aids in the construction of powerful malware analysis tools to combat malware that are becoming increasingly hard to analyze. Our goal is to provide a binary-instrumentation framework that is unobtrusive, portable, efficient, easy-to-use and reusable, supporting multithreading and SM-SC code, both in user- and kernel-mode. In this paper, we discuss the concept of *unobtrusive binary-instrumentation* and present the design, implementation and evaluation of SPiKE. We also illustrate the framework utility by describing our experience with a tool employing SPiKE to analyze a real world malware.

Keywords: Security, Malware, Instrumentation

1 Introduction

Malware analysis is a multi-step process combining both coarse- and fine-grained analysis schemes that provide insight into malware structure and functionality. Malware analysis environments (sandboxes) thus require various coarse- and fine-grained analysis tools to facilitate the development of an antidote. The first and a very important step in the analysis process, known as *Behaviour Monitoring*, involves monitoring malware behavior with respect to the system, to glean details which aid in further finer investigations. For example, the `W32.MyDoom` trojan and its variants propagate via e-mail and download and launch external programs using the network and registry. Such behavior, which includes the nature of information exchanged over the network, the registry keys used, the processes and files created etc., is inferred by employing coarse-grained analysis pertaining to process, network, registry, file and other related services of the host operating system (OS). Once such behavior is known, fine-grained analysis tools such as debuggers are employed on the identified areas to reveal finer details such as the polymorphic layers of the trojan, its data encryption and decryption engine, its memory layout etc.

Instrumentation – the ability to control constructs pertaining to any code – is a technique that is used for both coarse- and fine-grained analysis. Constructs can either be pure (functions following a standard calling convention) or arbitrary (code blocks composed of instructions not adhering to a standard calling convention). Control means access to a construct for purposes of possible semantic alteration. While instrumentation is trivial for OSs and associated applications that are open source, the task becomes complicated with deployments that are commercial and binary only. To this end there has been various research efforts which achieve instrumentation at the binary level. They fall into two broad categories: *Probe-based* and *JIT*. Probe-based methods such as Dtrace (Cantrill et al. 2004), Dyninst (Buck & Hollingsworth 2000), Detours (Hunt & Brubacher 1999) etc., achieve instrumentation by modifying the target construct in a fashion so as to enable a replacement function to be executed when the original construct is invoked. JIT methods such as Pin (Luk et al. 2005), DynamoRIO (Bruening 2004), Valgrind (Nethercote & Seward 2003) etc., on the other hand achieve instrumentation by employing a virtual machine (VM).

Both methods however fail in the context of malware. Probe-based methods are not transparent because original instructions in memory are overwritten by trampolines. Most if not all malware are sensitive to code modification and even instrument certain OS services for their functioning and stealthness. Also recent trends in malware show increasing anti-analysis methods, rendering probe-based schemes severely limited in their use. Current JIT schemes on the other hand, though transparent, do not contain support for multithreading and SM-SC code and are unable to analyze code executing in kernel-mode. Also JIT schemes are more suited for fine-grained instrumentation and are an overkill in terms of latency for coarse-grained analysis. This situation calls for a new coarse-grained instrumentation framework specifically tailored for malware analysis.

This paper discusses the concept of *unobtrusive binary-instrumentation*, and presents SPiKE, a realization of this concept that provides unobtrusive, portable, efficient, easy-to-use and reusable binary-instrumentation, supporting multithreading and SM-SC code in both user- and kernel-mode. The instrumentation deployed by SPiKE is *unobtrusive* in the sense that it is completely invisible and cannot be easily detected or countered. This is achieved by employing the virtual memory system coupled with subtle software techniques. The framework currently runs on OSs such as Windows (9x, NT, 2K and XP) and Linux on the IA-32 processors (Intel Corporation 2003) and is *portable* on any platform (processor and OS) that supports virtual memory. The framework achieves *efficient* instrumentation using special techniques such as redirection and localized-executions. SPiKEs API is simple yet powerful making the framework *easy-to-use.* Analysis tools are usually coded in C/C++ using SPiKEs API. The API is platform independent wherever possible allowing tool code to be *reusable* among different platforms, while allowing them to access platform specific details when necessary. To the best of our knowledge, SPiKE is the first coarse-grained binary-instrumentation framework that facilitates the construction of powerful malware analysis tools to combat malware that are increasingly becoming hard to analyze.

This paper is organized as follows: We begin by giving an overview of SPiKE's instrumentation mechanism in Section 2. We follow this with a detailed discussion on design and implementation issues in Section 3. In Section 4, we discuss our experience with one of our tools employing SPiKE to analyze a real world malware. We then evaluate in Section 5, the framework performance and compare it against other schemes. Finally, we consider related work on binary-instrumentation in Section 6 and conclude the paper in Section 7 summarizing our contributions with suggestions for future work.

2 Framework Overview

Instrumentation under SPiKE is facilitated by a technique that we call *Invisi-Drift.* The basic idea involves inserting what we call *drifters*, at memory locations corresponding to the code construct whose instrumentation is desired. Drifters are invisible logical elements that trigger an event when a read, write and/or execute occurs to the corresponding memory location. Each drifter has an associated *instrument*, a user-defined function to which control is transferred, when the drifter triggers. Instruments can then monitor and/or alter the executing code stream as desired. SPiKE's instrumentation strategy is stealth and cannot be detected or countered in any fashion. The framework also employs software techniques such as *redirection* and *localized-executions* to efficiently tackle issues involving reads, writes and/or executes to memory locations with drifters, thereby ensuring instrumentation presence while co-existing with other instrumentation strategies employed by a malware.

SPiKE relies on our stealth breakpoint framework, VAMPiRE (Vasudevan & Yerraballi 2005) to insert drifters at desired memory locations. Figure 1 illustrates the current architecture of SPiKE. The framework core consists of a code slice execute engine (CSXE) and an instrumentation API invoked by the analysis tool that uses SPiKE. The CSXE can be thought of as a pseudo-VM, that enables the framework to execute code fragments (slices) locally. This is needed to efficiently tackle various run-time issues involving SM-SC code (see Section 3.3) and invocation of the original construct at a drifter location (see Section 3.2). The CSXE employs a disassembler and a slice repository (acting as a cache) for its functioning. Since SPiKE's core resides in kernel-mode, it can capture both user- and kernel-mode code with ease.

As Figure 1 shows, there are typically three binary elements present during an analysis session: the target application, the analysis tool front-end, and the tool payloads. The tool payloads contain instruments that need to be in the target address space along with support code that deploys instrumentation. The target address space can be a new process, an existing process and/or the OS kernel itself. The tool front-end runs as a separate process and forms the user interface. The tool front-end initialises the framework and loads the tool payloads in the target address space, by a process that we call payloading (steps 1 and 2). The payloads upon gaining control, use the framework library to initialize SPiKE in the target address space and insert drifters at specified memory locations for instrumentation (step-3). When a drifter triggers (step-4), SPiKE transfers control to the associated instrument, which then does the necessary processing (step-5). Using SPiKE, the tool payloads can communicate with the tool front-end in real time. A point to be noted from Figure 1, is that, payloads in user-mode have a local copy of the CSXE linked to themselves. Thus, localized-executions occur at the same privilege level where the drifter was triggered. This ensures that malformed user-mode code cannot result in system instability, which would have been the case had localized executions always taken place in kernel-mode.

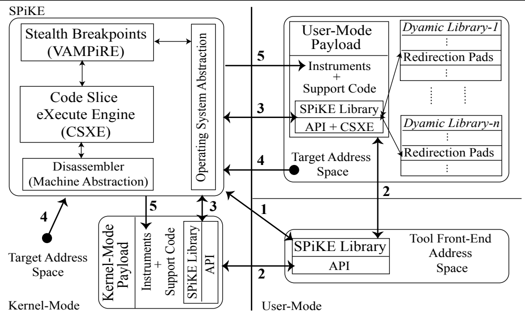


Figure 1: Architecture of SPiKE

SPiKE also employs a technique that we call *redirection*, for efficient instrumentation of functions housed in dynamic libraries. This is achieved by embedding code constructs known as redirection-pads into the target library. SPiKE is completely re-entrant, as it does not make use of any OS specific routines within the CSXE and uses shared memory with its own isolation primitives for interprocess communication. The SPiKE API is simple yet powerful to allow a tool writer to harness the complete power of the framework.

3 Design and Implementation

In this section, we present a detailed dicussion of SPiKE. We begin by discussing how the framework is deployed in the target execution space, followed by a detailed description of how the framework achieves instrumentation. Finally, we discuss the SPiKE API and some issues involving the framework stealthness.

3.1 Payloading

The analysis tool employing SPiKE, uses the framework API, to load the tool-payload(s) into the target execution space. The target execution space can be a new process, an existing process or the OS kernel itself. We call this process *payloading.* The design of our payloading scheme is unified and lends itself to implementation on a variety of OSs such as Windows, Linux, Solaris etc. Payloading is facilitated by the `ptrace` API on Unix systems and process and thread creation APIs such as `CreateProcess`, `OpenProcess`, `Suspend/ResumeThread` etc. on Windows systems. Using our payloading scheme allows us to attach to a new or an already running process in the same way as a debugger. For kernel-mode payloading, a tool-payload simply translates to a kernel-module for the specific OS. For user-mode payloading, a tool-payload translates to a dynamic library. Our payloading mechanism has the ability to track process dependencies (parent and child) and load the tool-payload(s) into newly created children processes automatically, thus aiding local (per process) and global (entire system) instrumentation. Further details regarding the payloading process and issues regarding OSs in the context of payloading can be found in our earlier work, SAKTHI (Vasudevan & Yerraballi 2004).

3.2 Invisi-Drift

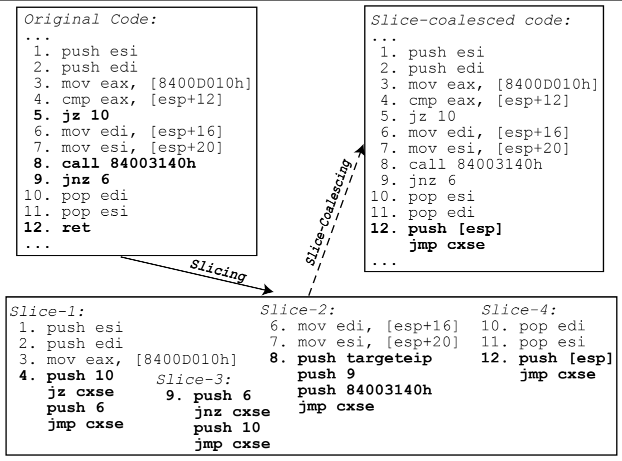
SPiKE inserts *drifters* at memory locations corresponding to the code construct whose instrumentation is desired. Drifters are logical elements that are a combination of a *code-breakpoint* and an *instrument*. A code-breakpoint provides the ability to stop execution of code at an arbitrary point during runtime and the instrument is a user-supplied function to which control is transferred when the breakpoint triggers. SPiKE makes use of our stealth breakpoint framework, VAMPiRE (Vasudevan & Yerraballi 2005) to set invisible code-breakpoints at memory locations corresponding to the code constructs whose instrumentation is desired. Stealth breakpoints provide unlimited number of invisible code, data and/or I/O breakpoints that cannot be detected or countered in any fashion. The basic idea involves exploiting the virtual memory system of the underlying platform by setting the attribute of the target memory page containing the memory location to *not-present* and using the page-fault exception (PFE) and subtle software techniques to trigger the corresponding breakpoint (Vasudevan & Yerraballi 2005).

A drifter under SPiKE can have a local (per-process) or global (entire system) presence and can be active or inactive at a given instance. A drifter can trigger (due to the triggering of the corresponding code-breakpoint) due to a read, write and/or execute to the memory location where it is inserted. When a drifter triggers, SPiKE invokes the instrument corresponding to the drifter, which can then monitor and/or alter program behavior by overwriting registers and/or memory (including the stack). They can also invoke the original code at the drifter location within themselves. This feature could be used to chain to the original construct to do the meat of the processing while employing changes to the result. As an example, let us consider the Windows OS kernel API `CreateProcess`. This API is responsible for creating new processes in the system. A possible instrumentation of this API might be used to keep track of the processes that are being created in the system and their relationships (parent/children). Under SPiKE, the instrument for the `CreateProcess` API can invoke the original API to create a process, perform its function (which is to keep track of the process hierarchy) and return to the caller. This model is much more flexible than a function prologue/epilogue instrumentation as found in probe-based systems such as DynInst (Buck & Hollingsworth 2000), DProbes (Moore 2001) etc. in that: (a) it decouples the instrument from the construct being instrumented, which enables the instrument to invoke the original construct as many times as required with different sets of parameters on the call stack and (b) it is a generic method that can capture invocations to constructs that are not pure. Instruments also have the ability to insert and/or remove drifters at any memory location dynamically.

3.3 Localized-Executions

With Invisi-Drift, when a drifter is inserted at a desired memory location, the entire memory page corresponding to the location, has its attribute set to *not-present* as described in the previous section. Further, for instrumentation persistence, such memory pages always need their attribute set to *not-present* for the lifetime of a drifter. These facts give rise to a couple of issues that need to be dealt with. The first is when the target code reads and/or writes to drifter locations or to locations within the memory page containing drifters. The second situation is when the original code at the drifter location is invoked from within the instrument. Both cases result in multiple PFEs due to the destination memory page attribute being *not-present*.

The framework employs the CSXE to eliminate such PFEs and execute such code with minimal latency. The CSXE *slices* code at a given location and stores it in a *slice-repository*. A slice is nothing but a straight line sequence of instructions that terminates in either of these conditions: (1) an unconditional control transfer, (2) a specified number of conditional control transfers or (3) a specified number of instructions. The slice-repository acts as a framework local cache and contains the collection of code sequences that have been sliced. Only code residing in the slice-repository is executed-never the original code and hence the term *localized-executions*.


```
Original Code:
...
 1. push esi
 2. push edi
 3. mov eax, [8400D010h]
 4. cmp eax, [esp+12]
 5. jz 10
 6. mov edi, [esp+16]
 7. mov esi, [esp+20]
 8. call 84003140h
 9. jnz 6
10. pop edi
11. pop esi
12. ret
...
```

```
Slice-coalesced code:
...
 1. push esi
 2. push edi
 3. mov eax, [8400D010h]
 4. cmp eax, [esp+12]
 5. jz 10
 6. mov edi, [esp+16]
 7. mov esi, [esp+20]
 8. call 84003140h
 9. jnz 6
10. pop esi
11. pop edi
12. push [esp]
    jmp cxse
...
```

```
Slice-1:
 1. push esi
 2. push edi
 3. mov eax, [8400D010h]
 4. push 10
    jz cxse
    push 6
    jmp cxse

Slice-3:
 9. push 6
    jnz cxse
    push 10
    jmp cxse

Slice-2:
 6. mov edi, [esp+16]
 7. mov esi, [esp+20]
 8. push targeteip
    push 9
    push 84003140h
    jmp cxse

Slice-4:
10. pop edi
11. pop esi
12. push [esp]
    jmp cxse
```

Figure 2: Localized-Executions using CSXE

Figure 3 illustrates a typical slicing process. A slice consists of one or more exit sequences which transfer control to the CSXE. The CSXE determines the target address, performs slicing for the target if it has not done before, and resumes execution at the target slice. Branches to and from the CSXE result in the saving of the current registers and flags. Generating exit stubs for unconditional control transfers is as simple as performing an unconditional jump into the CSXE. For certain unconditionals such as the `call` instruction, the stub pushes the destination instruction pointer (targeteip, line 8, slice-2, figure 2) and performs an unconditional jump to the CSXE. This ensures that code employing position independent access (as seen in many SM-SC code) works without any problems. For unconditional brances, the CSXE employs a host of exit stubs for different variants (slice-1 and slice-4, figure 2). The basic idea being to translate a conditional into a conditional and an explicit branch. This model enables *slice-coalescing*, whereby a group of slices can be coalesced to minimize exits to the CSXE. Slice-coalescing is a powerful mechanism that can produce code fragments which are very similar to the original code that can be locally executed with minimal latency. The CSXE also stores what are known as *ghosts* for read and/or writes to memory pages containing drifters. These are duplicates of the original memory page where drifters are inserted and are used when the original code at the drifter location need to be invoked within the instrument. Ghosts allow others to install their own instruments while preserving the original function for invocation by the framework instruments.

The CSXE differs from traditional VMs employed in current JIT schemes in that it does not reallocate registers or perform any liveliness analysis etc. This eliminates the latency associated with dynamic compilation since localized execution is typically over a limited code fragment (maximum of a page size). The CSXE stops slicing if the target address of goes beyond the memory page containing the drifter for instances where the code residing at the drifter location needs to be invoked from the instrument. For situations involving reads and/or writes to drifter locations or locations within the memory page of the drifter, the CSXE stops code slicing after a certain number of instructions that are obtained and refined over time by employing simple heuristics on the target code. Eg. A sequence of writes to a given location, often happen with a loop over a specified number of instructions. The fact that the CSXE is completely re-entrant, does not re-allocate registers, does not tamper with the code stack and employs subtle exit stubs for conditional and unconditionals, endows it with the capability to support multithreading and SM-SC code — features that are unavailable in current JIT schemes.

3.4 Redirection

SPiKE allows drifters to be inserted at arbitrary memory locations for instrumentation purposes. However, this direct method of instrumentation incurs a latency when it comes to invoking the code at the drifter location from within the instrument due to localized-executions as described in the previous section. This latency can be nullified if we hinge on the fact that most of the services offered by the OS or an application are in the form of dynamic link libraries (DLL) or shared libraries. For example, the socket `send` function is provided as an export of the DLL, `WINSOCK.DLL` under the Windows OSs and as an export of the shared library, `libsocket.so` under the Linux OS. While there are programs that make use of static libraries, their numbers are very limited (about 1% of all applications in our experiment). Also on some OSs such as Windows it is simply not possible to statically link against the system libraries without severely compromising compatibility.

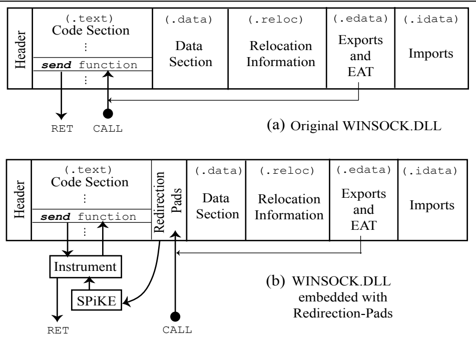


Figure 3: Redirection for Dynamic Libraries

Every DLL or a shared library exports the functions contained within it using an *export-table*. Among other fields, the export-table contains an array of pointers to functions within the library known as the *Export Address Table* (EAT). When an application links with a DLL or a shared library, it *imports* a set of functions from it. This is in the form of an *Import Address table* (IAT) or a *Procedure Linkage Table* (PLT) within the application. Further details regarding the executable formats of DLLs and shared libraries can be found in the Portable Executable Specification (Microsoft Corporation 2004) and the Executable and Link Format Specification (Unix System Laboratories 1998). The OS loader, obtains a list of the functions that are being imported, loads the appropriate libraries, fetches the addresses of the functions using the EAT and patches the IAT or the PLT within the application. Figure 3a shows a sample function invocation from an application using the DLL, `WINSOCK.DLL`.

SPiKE re-writes the DLL or the shared library that contains the function(s) to be instrumented, with redirection pads. The re-writing is carried out in binary and does not require the sources and/or any debug symbols for the library. Every function in the module, that is intended to be instrumented will have a unique *redirection-pad* embedded within the module. A redirection-pad is a sequence of instructions which finally transfer control to the function that is associated with it. The export-table entries of the module are then overwritten to point to the redirection pads of the corresponding functions. Thus, when an application links to the library and imports a function which has a redirection-pad, it will in reality invoke the redirection-pad code instead of the original function, when the target library is loaded and the OS loader performs the IAT patch. However, the semantics of the function invocation will remain the same as the redirection-pads chain to the original function without any changes. Figure 3b shows the same function invocation for the DLL, `WINSOCK.DLL` with the redirection-pads in place. Note from the figure that the redirection-pads are embedded directly in the code section of the library and not as a separate section. This is needed to hide the framework from the malware being analyzed (see Section 3.6).

Thus, when drifters are inserted at memory locations corresponding to the functions within the DLL or shared libraries, they will be inserted on memory pages containing the redirection pads. In other words, the original function is available for invocation from within the instrument with no latency involved. The redirection-pads however, still incur a latency due to reads and/or writes but as a side-effect elminate the need for ghost pages, since any modifications to the original code is in fact modifying the redirection-pad code. The instructions generated in the redirection-pads are polymorphic to prevent the malware from detecting the framework using pattern-based schemes on the redirection-pads (see Section 3.6). The framework also allows the user to tune the number of redirection-pads per memory page, and the range for the number of instructions that will be generated for a redirection-pad.

3.5 SPiKE API

Anaylsis tools that employ SPiKE are usually written in C/C++ using SPiKE's API. The API is easy-to-use and is designed to be platform independent whenever possible allowing tool code to be re-usable while allowing them to acess platform specific details when necessary. In Figure 4, we list a partial code of one our preliminary tool employing SPiKE under the Windows OS. The tool, `DrvDllMon`, enables complete monitoring of drivers and dynamic link libraries that are being loaded and unloaded in the system.

The main interface to SPiKE is provided in the form of five easy to use API functions, `spike_init`, `spike_payload`, `spike_insertdrifter`, `spike_removedrifter` and `spike_comm`. As seen from Figure 4, a tool using SPiKE is typically composed of two elements. A tool payload and a tool front-end. The tool payload contains the instruments that need to be in the target address space along with support code that deploys instrumentation. The tool front-end forms the user interface. The front-end of the tool initializes the framework and typically invokes the `spike_payload` API to load the payload into the target address space. The `spike_init` API is used to initialize the framework. The payload upon gaining control, initializes the framework in the target address space, and uses the `spike_insertdrifter` and `spike_removedrifter` API calls to insert or remove drifters at selected memory locations. As mentioned earlier, drifters can have a global (`DGLOB`) or local presence (`DLOCL`) with the ability to trigger on a read (`DREAD`), write (`DWRITE`) and/or execute (`DEXEC`) to its corresponding memory location. Communication between the payload and the tool front-end occurs via SPiKEs API `spike_comm`.

Note that the payload code is automatically translated into the appropriate executable during compile time, depending on whether instrumentation is desired in user- and/or kernel-mode. The instruments access relevant information via the argument of type `DRIFTERINFO`, which among other fields includes a pointer to the current stack and a function pointer that can be used to invoke the original construct at the drifter location from the instrument. The instruments can thus alter program registers, memory (including the stack) and perform any semantic changes. The instruments can also access instrument specific data using the `exinfo` field of the argument. This feature, for example allows a single instrument to be associated with multiple drifters as seen from Figure 4 where the instrument `LL_Inst` is associated with drifters to the ANSI (`LoadLibraryA`) and UNICODE (`LoadLibraryW`) versions of the API `LoadLibrary`, which is used to load a DLL under the Windows OS. SPiKE also provides a host of other support APIs apart from the ones discussed above, related to architecture and OS specific elements such as instructions, processes, libraries, files etc.

```
---drvdllmonp.c (drvdllmon payload)---
#include <spike.h>
...
void LL_Inst(DRIFTERINFO *di){ \\LoadLibrary API Instrument
 ...
 if(di->exinfo[0] == 'W') \\UNICODE version
  spike_comm("0x%X: LoadLibW(%ls)", di->stack[0], di->stack[4]);
  ...
 di->retval= di->originalcode(di->stack[4], di->stack[8]);
}

void payload_init(PROCESSINFO *p){ \\payload initialise function
 ...
 spike_init(); \\initialise framework in process address space
 ...
 \\instrument the ANSI and UNICODE version of Loadlibrary API
 addr1=spike_addr("kernel32.dll", "LoadLibraryA");
 addr2=spike_addr("kernel32.dll", "LoadLibraryW");
 spike_insertdrifter(addr1, DEXEC | DGLOB, LL_Inst, 'A');
 spike_insertdrifter(addr2, DEXEC | DGLOB, LL_Inst, 'W');
 ...
}
...
---drvdllmonf.c (drvdllmon front-end)---
#include <spike.h>
...
int main(){
 ...
 spike_init(); \\initialise framework
 ...
 spike_payload(pid, "drvdllmonp"); \\load the payload
 ...
}
...
```

Figure 4: DrvDllMon, a tool employing SPiKE

3.6 Stealth Techniques

SPiKE uses drifters, CSXE and redirection-pads for implementing un-obtrusive instrumentation. However, these elements and the latency that they introduce can be detected during runtime albeit with some subtle tricks. We now present some simple techniques that the framework employs for stealthness to counter such methods. While there are a host of detection methods and their antidotes, due to space constriants, we will only concentrate on a few important ones.

Drifters rely on the virtual memory system for their operation. However, drifter triggerings result in increased latency due to PFEs being invoked that is not present during normal execution. A malware could use this fact to detect if its being analyzed. As an example, on the IA-32, a malware could use the `RDTSC` instruction (returns the number of clock cycles that have elapsed since system-bootup) and/or the real-time clock to obtain a relative measure of its code fragment execution time. Depending on the system SPiKE is run under, the framework applies a clock patch resetting the time stamp counter and/or the real-time clock to mimic a value close to that of a normal execution.

SPiKE makes use of redirection-pads within the dynamic libraries to void latency issues during invocation of original constructs from the instruments. The redirection pads are embedded within the library in the same section as the function code. This is necessary since, if the pads were embedded in a separate section, a malware could check the EAT entries and compare them with the library header to detect that the EAT entries have been routed. However, since the redirection-pads are within the code section of the library, there is no way of distinguishing them from the library functions. The redirection pad code is polymorphic in the sense that one cannot employ any signature detection to check for them. The framework employs a polymorphic code generator, that allows fine tuning of the upper-bound of the number of instructions that will be generated in a redirection-pad. In other words, a malware cannot search for a unique instruction within the redirection pad code which would establish a postive detection of the framework.

SPiKE employs the CSXE for localized code execution during read, writes or executes to drifter locations. Though the CSxE supports SM-SC code, a malware could use a technique such as to single-step its own instruction stream and performing the actual operation within its single-step handler. The malware single-step handler can check the return address on the exception stack and figure that the code is not being executed at the address it should have been (since localized-executions execute the slice from a different memory location altogether). The CSXE can recognize such situations, and will execute the code slice instructions one by one invoking the target single-step handler with the original instruction addresses.

The framework also employs a polymorphic engine to ensure that every instance of its deployment is different in the form of any privileged modules, environment variables, configuration files and code streams. Thus, no malware can detect SPiKE by searching these elements for a pattern.

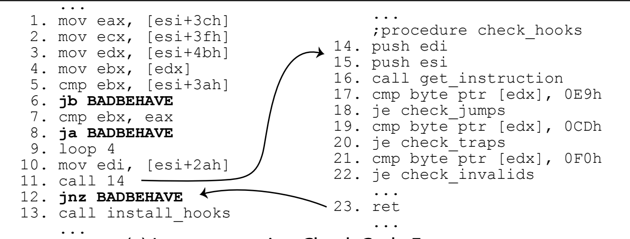
4 Experience

In this section we discuss our experience with one of our tools employing SPiKE to analyze a real world malware, therby illustrating the framework utility. Our tool called `WatchDog`, currently runs under the Windows OS, and enables monitoring of various user- and kernel-mode components of the OS, offering insight into malware behavior with respect to the system. `WatchDog` is a simple yet versatile tool that allows real-time monitoring of Windows APIs related to files, processes, registry, network, memory and various other sections of the OS both in user- and kernel-mode. It enables complete monitoring of drivers and dynamic link libraries that are being loaded and unloaded in the system. The tool also allows instrumenting the entry and exit points of the dynamic link libraries and drivers with support for EAT access monitoring, critical data access monitoring, dependencies, per process filters and a host of other features which make it an indispensable tool to monitor malware behavior in a system. `WatchDog` is one of the many components that make up our malware analysis environment that is currently under research and development. This section will discuss our experience using `WatchDog` on several monitoring sessions with a Windows based trojan, `W32.MyDoom`. We chose `W32.MyDoom` as our candidate for discussion, since it has several variants, employing a variety of anti-analysis tricks that one would typically encounter in recent malware.

The `W32.MyDoom`, with variants commonly known as `W32.MyDoom.X-mm` (where X can be S, G, M etc.) is a multistage malware. It generally arrives via an executable e-mail. Once infected, the malware will gather e-mail addresses, and send out copies of itself using its own SMTP engine. It also downloads and installs backdoors and listens for commands. Once installed, the backdoor may disable antivirus and security software, and other services. The downloaded trojan might allow external users to relay mail through random ports, and to use the victim's machine as an HTTP proxy. The trojans downloaded by the `W32.MyDoom` envelope, generally posses the ability to uninstall or update themselves, and to download files.

The `W32.MyDoom` and modified strains cannot be completely analysed using traditional JIT and/or probe-based binary-instrumentation. The malware employs a polymorphic code envelope and employs efficient anti-probe techniques to detect probes and will remain dormant and/or create system instability in such cases. For the purposes of discussion, we look at a simplified code fragments of different variants of the `W32.MyDoom` under monitoring sessions with `WatchDog` and our malware analysis environment. The fragments are shown in the

IA-32 assembly language syntax. We have removed details from the code that are not pertinent to our discussion and have restructured the code for easy understanding. Consider a fragment of code shown in Figure 5a.

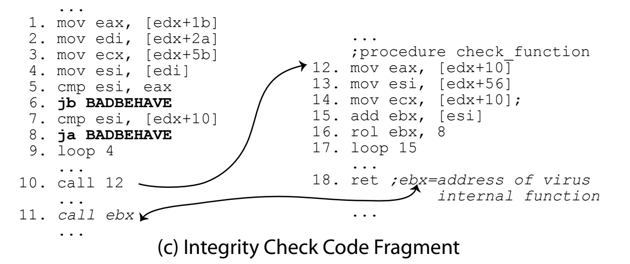


```
     ...
 1. mov eax, [esi+3ch]              ...
 2. mov ecx, [esi+3fh]              ;procedure check_hooks
 3. mov edx, [esi+4bh]          14. push edi
 4. mov ebx, [edx]              15. push esi
 5. cmp ebx, [esi+3ah]          16. call get_instruction
 6. jb BADBEHAVE                17. cmp byte ptr [edx], 0E9h
 7. cmp ebx, eax                18. je check_jumps
 8. ja BADBEHAVE                19. cmp byte ptr [edx], 0CDh
 9. loop 4                      20. je check_traps
10. mov edi, [esi+2ah]          21. cmp byte ptr [edx], 0F0h
11. call 14                     22. je check_invalids
12. jnz BADBEHAVE                   ...
13. call install_hooks          23. ret
     ...                            ...
```

(a) Instrumentation Check Code Fragment

```
......
CS:112050DCh CreateFile("%SYSTEMROOT%\WINVPN32.EXE", ...) = F124h
......
CS:0042312Ah winsock.dll!send(1A20h, [DS:00404050h], ...) = 800h
CS:11204A2Bh a32ss32.dll!send(1A20h, [DS:00404050h], ...) = 800h
CS:112051E0h a32ss32.dll!connect(213Bh, 84.168.21.11, 200) = 0h
......
CS:0044A340h winsock.dll!send(1A20h, [DS:00404010h], ...) = 400h
CS:11204A2Bh a32ss32.dll!send(1A20h, [DS:00404010h], ...) = 400h
CS:11205AE4h a32ss32.dll!send(213Bh, [DS:1121A000h], ...) = C00h
CS:11206120h a32ss32.dll!recv(213Bh, [DS:12005000h], ...) = 200h
CS:11206120h a32ss32.dll!recv(213Bh, [DS:12102000h], ...) = 200h
......
CS:1120AE70h WriteFile(F124h, [DS:12005000h], ...) = 1000h
CS:1120AE70h WriteFile(F124h, [DS:12102000h], ...) = 1000h
......
```

(b) Behavior Log showing a Trojan Download



```
     ...
 1. mov eax, [edx+1b]
 2. mov edi, [edx+2a]               ...
 3. mov ecx, [edx+5b]               ;procedure check_function
 4. mov esi, [edi]              12. mov eax, [edx+10]
 5. cmp esi, eax                13. mov esi, [edx+56]
 6. jb BADBEHAVE                14. mov ecx, [edx+10];
 7. cmp esi, [edx+10]           15. add ebx, [esi]
 8. ja BADBEHAVE                16. rol ebx, 8
 9. loop 4                      17. loop 15
     ...                            ...
10. call 12                     18. ret ;ebx=address of virus
     ...                                  internal function
11. call ebx                        ...
     ...
```

(c) Integrity Check Code Fragment

Figure 5: (a)-(c) W32.MyDoom Anti-analysis Tricks and Behavior Log

The W32.MyDoom and its variants instrument (hook) several APIs within the system. One such API that is hooked is the ZwQueryDirectoryFile. The ZwQueryDirectoryFile API under the Windows OSs (NT, 2K and XP), is a kernel function that is used to obtain the list of files within a particular directory. Calls to upper level file functions are ultimately translated to invoke this API within the kernel. The malware hooks this API, so that it modifies the return query buffer in a way that would prevent any regular application from seeing its critical files. There are several ways to instrument this API. The first method is by changing the pointers in the system service table pointed to by KiSystemServiceDispatcherTable to point to the instrumented functions. The pointer location for a particular system service can be found out by disassembling the dynamic library, NTDLL.DLL to get the indices for several system-calls. The second method is by using a probe-based instrumentation on the starting address of the API. Thus, if one instruments the API before-hand, using either of the two methods, it is easy to observe the behaviour of the malware, even if it hooks the API once again (since the malware will have to invoke the original API to obtain the populated query buffer in the first place).

However, the W32.MyDoom is intelligent to spot this. It performs a couple of checks to see if the API has been already hooked. The first check is a detection of a system-table hook (lines 1-9, Figure 5a). The malware uses a fact that original system-table entries point to code within NTOSKRNL.EXE, the Windows OS kernel. Thus, if one were to change the system-table entries to point to their own code, it will be at an address that is outside the image-space of NTOSKRNL.EXE. If this detection passes, the malware jumps to a location within its code which leads to a dormant operation or in certain cases causes system instability.

The second form of detection is for probe-based schemes at the target API. All probe-based schemes rely on the use of a unconditional transfer instruction (branch, trap etc.) at the address of the target code to be instrumented. This makes them an easy prey to detection. As seen from lines 14-22 of procedure check_hooks of Figure 5a, the malware checks to see if the code at the target address is any instruction that could potentially transfer control to an instrument (jump, trap, invalid instructions etc.). The check sweeps through the target API address checking for such instructions before a conditional is encountered. Since current probe-based frameworks rely on the use of control transfer instructions at the start of the target function thats being instrumented, they are rendered unusable in this context. The idea of inserting the probe, after the malware has inserted its hook doesnt accomplish any functionality since it would never see the unmodified buffer, but is also defeated since there are several checks, scattered throughout the code of W32.MyDoom, within its polymorphic layers, to ensure that no other system can hook the APIs after it has.

However, using SPiKE it is trivial to instrument such APIs, while at the same time allowing the malware hook to be active. In effect, WatchDog bypasses the checks and can monitor such APIs in a stealth fashion. Figure 5b shows a sample log from the utility revealing a trojan download and implant.

Certain variants of the W32.MyDoom use localized DLL's for their operation. For example, the trojan renames WINSOCK.DLL, the dynamic library associated with socket functions, to a random name and replaces it with its own DLL instead. The replaced DLL is coded in a fashion employing polymorphic layers and incorporates malware functionality while finally chaining to the functions within the renamed WINSOCK.DLL. As seen from the log of Figure 5b, WatchDog captures both the replaced (WINSOCK.DLL) and the renamed (A32SS32.DLL) dynamic library invocations, allowing us to spot an activity such as a trojan download and implantation as shown. The tool enables logging of several important pieces of a function invocation such as the parameters, the return value, the location it was invoked from, the stack and memory contents etc. WatchDog also features what we call *selective logging*, whereby only invocations arising from a specified range of memory is monitored. This is a very useful technique that helps maintain the clarity of information that is logged. The tool also features a script based interface allowing users to add real-time actions to monitored events.

The localized DLL's of W32.MyDoom employ certain anti-probing tricks. We noticed a couple of such tricks in one variant during our analysis. The code fragment of the malware in this context is shown in Figure 5c. The malware employs integrity checks using checksums on the socket primitives on the replaced WINSOCK.DLL (check_function procedure, Figure 5c). Thus, inserting traditional probes on such functions, results in erratic behavior (lines 6 and 8, Figure 5c). Solutions employing replacing the replaced DLL or rehousing the EAT entries in the replaced WINSOCK.DLL are defeated by similar checksum fragments that are embedded within the malware code. If the malware detects a malformed replaced DLL, it will overwrite it with a new copy. As an added detection, the malware also checks for probe insertions in the renamed DLL (A32SS32.DLL) once loaded. Manual patching of such integrity checks are cumbersome since: (a) many such fragments are scattered throughout the malware code and (b) the checksum themselves are used as representatives of the target address of a call that performs some processing pertaining to the malware functionality (line 18 and 11, Figure 5c). Thus, on a valid checksum the call performs the desired internal function whereas on an incorrect checksum, it branches to a location where the code is nothing but garbage. With SPiKE, instrumenting functions within the renamed DLL is trivial using redirection-pads. For instrumentation of the replaced DLL, drifters are inserted at desired functions directly.

As seen, traditional probe-based insturmentation methods do not suffice to study code employing self-modification, self-checking, hooking and/or any form of anti-analysis and/or anti-debugging schemes as in the case of the `W32.MyDoom` and other similar malware. With SPiKE however, this task is greatly simplified. The framework allows instrumentation of the target code without any form of code modification in a stealth fashion making it very hard to detect and counter. The latency of the framework is well suited for interactive monitoring (as seen from its performance measurements in the next section). These features combined with the fact that the recent trend in malware has been to employ schemes to detect and prevent any form of analysis, make SPiKE the first and a very powerful instrumentation framework to aid in the construction of malware analysis tools.

5 Performance Evaluation

In this section, we first report the performance of SPiKE without any active instrumentation. We then report the performance of SPiKE's direct (without redirection) and redirection-based instrumentation methods. Finally, we compare SPiKE with some widely used JIT and probe-based tools, and show that the framework incurs low latencies while providing features that are highly conducive to malware analysis.

Before we proceed to discuss the performance measurements of the framework, a few words regarding the test-bench are in order. The current version of SPiKE supports IA-32 (and compatible) processors running the Windows and Linux OSs. Our experimental setup consists of, an AMD Athlon XP 1.8 GHz processor with 512MB of RAM, running Windows XP and Windows 2000, and an Intel Xeon 1.7 Ghz processor with 512MB of RAM running Linux. To validate SPiKE, we use our analysis tool `WatchDog` (see Section 4). We use processor clock cycles as the performance metric for our measurements. This metric is chosen, as it does not vary across processor speeds and also since it is a standard in literature related to micro-benchmarks. The `RDTSC` instruction is used to measure the clock cycles. Unless otherwise specified, for the graphs encountered in the following sub-sections, the x-axis is the amount of extra clock cycles that are incurred as opposed to the native run-time of the function and the latency is measured by executing sufficient runs and averaging the values to obtain a confidence interval of 95%. Also, parts of some graphs are magnified (indicated by dotted lines) to provide a clear representation of categories with low values on the x-axis.

5.1 Latency Without Instrumentation

SPiKE has zero instrumentation effect for each instrumentation point for inactive direct instrumentation. This is because, a drifter inserted at a memory location can only trigger when the corresponding memory page attribute is set to *not-present* upon drifter activation. However, SPiKE's redirection-based instrumentation incurs a non-zero, but low run-time overhead when instrumentation is inactive. This is due to the execution of the instructions in the redirection-pads corresponding to the functions to be instrumented. The number of these instructions are variable and depend upon a user-defined range for a session (see Section 3.4). The framework latency for various dynamic library functions, from different analysis sessions under the Windows OSs, for three arbitrary user-defined ranges are as shown in Figure 6a.

The average latency for the functions when using the user-defined ranges shown in Figure 6a, are 343, 1405 and 2910 clock cycles respectively. For a 1.8 GHz processor, these numbers average to .19μs, 0.78μs and 1.6μs respectively, which is minimal. In general, the more the number of instructions per redirection-pad, the more is the latency and the less is the chance of its detection. In our opinion a user-defined range of 15–50 instructions should suffice to keep the redirection-pads from being detected using any patterns. A point to be noted from the graph is that, for the same function (`NtReadFile(1)` in Figure 6a for example), one could have different latencies per analysis session for the same user-defined range. This is because of the polymorphic nature of code that is generated in its redirection-pad.

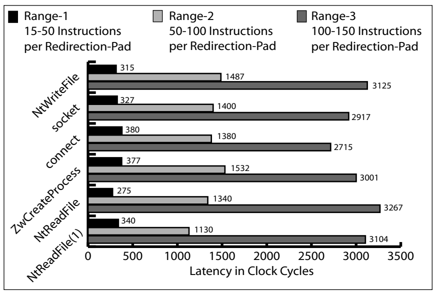
5.2 Latency With Instrumentation

We now study the performance of SPiKE with active direct and redirection-based instrumentation. We divide the total run-time overhead into three components: (a) latency due to instrument invocation, (b) latency due to invocation of the original construct at a drifter location from the instrument, and (c) latency due to reads, writes and/or executes to a memory page containing active drifters.

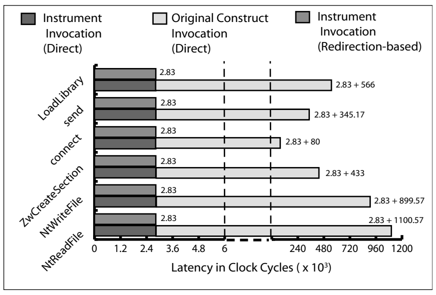
The instrument invocation time is the time that has elapsed after the transfer of control to a code construct (control transfer is usually done via a `call` for a code construct that is a function), and before control is handed over to the instrument. Figure 6b shows the latency involved in invoking the instrument and the original construct at a drifter location, for both direct and redirection-based instrumentation of SPiKE, for various arbitrary functions within DLLs and shared libraries under the Windows and Linux OSs. As seen from Figure 6b, both direct and redirection-based methods incur the same instrument invocation latency. This is due drifter triggering, which incurs a constant processing overhead for any instrumented memory location due to the invocation of the PFE (see Section 3.2). However, considering the latency involved in invoking the original construct at a drifter location, the direct method incurs a higher performance penalty when compared to the redirection-based method. This is due to the framework employing the CSXE for localized-executions in the direct method. In contrast, the redirection-based method uses redirection-pads to eliminate this overhead.

The framework also incurs other forms of latency in the form of reads, writes and/or executes to a memory page containing active drifters. For example, a malware might try to install its own instrumentation at a memory location where a drifter is inserted. This would result in PFEs due to reads and/or writes to the drifter location. Another example would be when a drifter is inserted at the start of a function, but the memory page containing the function also houses parts of another function that is not instrumented. Thus, executes to the uninstrumented function also trigger PFEs. SPiKE employs the CSXE to tackle such cases which results in the aforementioned latency. Coming up with representative inputs for such cases, for purposes of performance evaluation, is not an easy task since they depend on a lot of factors, chief among them being the method of analysis adopted by an individual, the structure of the executing code and dependency of one function over the other, which are not easily characterized. Thus, we will concentrate on presenting the performance of the framework related to some specific code fragments for such cases, and show that the framework latency is within limits to suit interactive analysis. The performance of the framework for other situations can be estimated in a similar fashion.

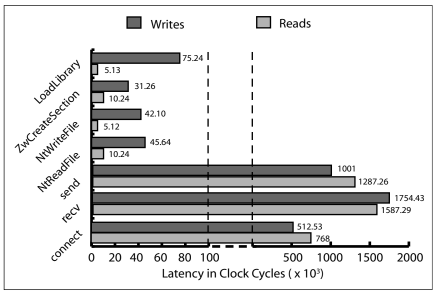
Consider the code fragment in Figure 5a. This fragment of the `W32.MyDoom` trojan checks to see if a certain group of system functions have been instrumented prior to its execution (see Section 4). As seen from Figure 5a, the procedure `check_hooks` reads every instruction from the memory locations corresponding to the func-



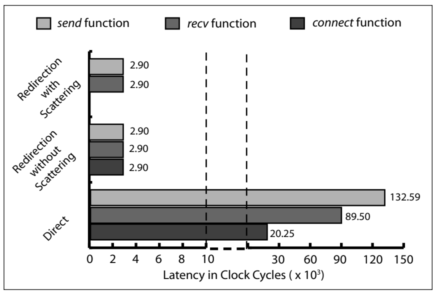
Latency Without Instrumentation
(a)



Latency of Instrument and Original Construct Invocation
(b)



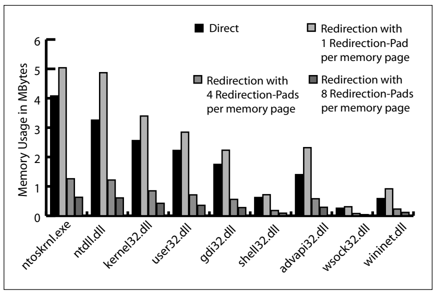
Latency of Reads and Writes to Memory Page with Active Drifters
(c)



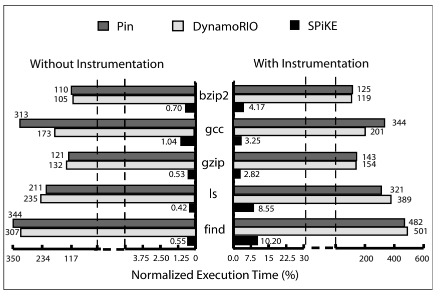
Latency of Executes to Memory Page with Active Drifters
(d)

Feature / Framework	Transparent	Original Construct Invocation	Success Rate	Kernel-Mode Support	Stealth	Multithreading	Arbitrary Code	SM/SC Code	Platform Support
DProbes	✗	✗	< 100%	✓	✗	✓	✗	✗	Linux/IA-32, IA-64
Detours	✗	✓	< 100%	✓	✗	✓	✗	✗	Windows/ IA-32
DynInst	✗	✗	< 100%	✓	✗	✓	✗	✗	Windows, Linux/ IA-32
Pin	✓	✗	100%	✗	✗	✗	✓	✗	Linux/ IA-32, IA-64 EMT64, ARM
DynamoRIO	✓	✗	100%	✗	✗	✗	✓	✗	Windows, Linux/ IA-32
SPiKE	✓	✓	100%	✓	✓	✓	✓	✓	Windows, Linux/ IA-32

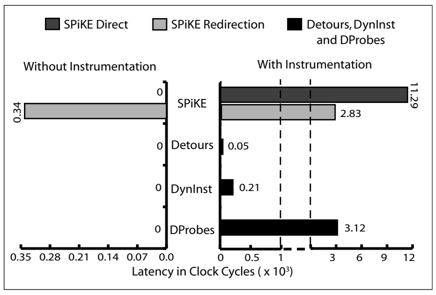
Feature Comparison
(e)



Memory Consumption
(f)



Performance Comparison with JIT Frameworks
(g)



Performance Comparison with Probe-based Frameworks
(h)

Figure 6: (a)-(h) SPiKE's Quantitative and Qualitative Evaluations

tions, checking for inserted probes. If this check passes, the virus uses the procedure `install_hooks` to write out its own probes at desired memory locations. Figure 6c shows the latency incurred by the framework due to such reads and writes on certain system functions checked and instrumented by the procedures `check_hooks` and `install_hooks` and to functions housed in localized DLLs employed by the virus (in this case `WINSOCK.DLL`), using both direct and redirection based instrumentation. One can observe from the graph that for functions housed in system and/or standard libraries, the number of reads and/or writes is very minimal (mostly the first few instruction of the function), whereas for a malware specific DLL that overwrites its own code, the latency is higher on account of multiple reads and/or writes due to its polymorphic nature. In either cases, the latency is well within the limits to suit interactive monitoring and analysis.

Now, let us consider the behavior log shown in Figure 5b, that reveals a trojan download (see Section 4). Let us consider the APIs `send`, `recv` and `connect` of the dynamic library `A32SS32.DLL` and assume that only `send` and `recv` are instrumented using SPiKE. Figure 6d shows the latency incurred (instrument invocation and original construct invocation put together) for both direct and redirection-based schemes of instrumentation, for a single invocation of the three APIs. As seen, in the direct method, the function `connect` incurs an overhead even though it is not involved in instrumentation. This is due to the fact that APIs `send` and `connect` share the same memory page in `AS23SS32.DLL`. Thus, the framework needs to employ localized-executions using the CSXE for the `connect` function which explains the latency.

However, when it comes to the redirection-based method, the latency of the framework depends upon the number of redirectors per memory page and the way the redirection pads are chosen for the functions. As an example consider the graph in Figure 6d which shows the latency of the three APIs, with 4 redirection pads per memory page, and assigned redirection-pads that fall on the same memory page. This results in a higher runtime latency, since APIs that are not involved in instrumentation, but lie within the same memory-page incur the overhead of localized-executions. However, by carefully assigning redirection-pads for the functions such that they do not fall on the same memory page, the latency can be reduced. This is shown in Figure 6d, with 4 redirectors per memory page, but with the APIs assigned redirection pads in such a fashion so that they do not occupy the same memory page. This technique, that we call *scattering*, can be applied to functions housed in DLLs and/or shared libraries and helps in reducing the latency involved in executing functions that are not involved in instrumentation but share the same memory page with functions that are instrumented.

5.3 Memory Consumption

The direct and redirection based instrumentation strategies of SPiKE incur a memory overhead due to their design elements. The direct method has zero memory overhead when instrumentation is inactive. For active instrumentation using the direct method, every memory page containing a drifter incurs a memory overhead of one extra memory page. This extra memory page, called a *ghost*, is used by the framework CSXE for localized-executions (see Section 3.3). The redirection-based method on the other hand incurs a memory overhead when instrumentation is both active or inactive. This is due to the redirection-pads being embedded in the DLLs and/or shared libraries which consume memory irrespective of any instrumentation.

The exact memory overhead for the direct method, depends on the number of drifters and the memory locations at which they are inserted. Similarly, for the redirection-based method, the actual memory overhead depends on the number of redirection-pads per memory page and the number of functions that are instrumented in a DLL and/or shared library. These factors depend on the nature of analysis employed by an individual which as mentioned before is not easy to characterize. Thus, we will concentrate on presenting the memory overhead of the framework in the context of a specific analysis session under the Windows OS. The memory overhead of the framework for other situations can be estimated in a similar fashion.

Figure 6f shows the memory overhead for both direct and redirection-based instrumentation methods of SPiKE, for selected dynamic libraries, from a session used to analyse the `W32.MyDoom` trojan under the Windows OS (see Section 4). Readings were obtained by instrumenting all the exported functions within each DLL and/or shared library using the direct method as well as the redirection-based method with arbitrary number of redirection pads per memory page. As seen from Figure 6f, the worst-case memory consumption (all functions instrumented within all selected libraries with 1 redirection-pad per memory page) is around 23MB which is not very demanding. A point to be noted from the graph in Figure 6f is that, the higher the number of redirection-pads per memory page, the lesser is the memory consumed. However, choosing too high a value might have effects on techniques such as *scattering* and the framework stealthness. In our opinion, a value of 4 to 16 for the number of redirection-pads per memory page suffices to keep the framework memory utilization minimal while ensuring its efficacy.

5.4 Framework Comparison

We now compare SPiKE against some popular JIT and probe-based instrumentation frameworks such as Pin, DynamoRIO, Detours, Dyninst and DProbes on the IA-32 (and compatible) processors. Detours runs exclusively on the Windows OS, DProbes and Pin run exclusively on the Linux OS, while DynamoRIO and DynInst run on both OSs. We used the latest release of each framework for this experiment: Pin Kit 2411 (Luk et al. 2005), DynamoRIO 0.9.4 (Bruening 2004), Detours 2.0 (Hunt & Brubacher 1999), Dyninst 4.2.1 (Buck & Hollingsworth 2000) and DProbes 2.6.9 (Moore 2001). The first part of this section discusses the features provided by these JIT and probe-based instrumentation frameworks and how SPiKE compares qualitatively. A quantitative comparison in the later part sheds light on the overhead involved in applying SPiKE's instrumentation when compared to the other frameworks.

Figure 6e shows various attributes of an instrumentation framework and how SPiKE compares qualitatively to various JIT and probe-based frameworks. As seen, SPiKE though standing out in features supported in the context of malware (transparent, stealth, multithreading, SM-SC code and arbitrary code), also offers general capabilties that match (in certain cases better, such as original construct invocation, kernel-mode support and success-rate) that of the existing frameworks.

For a quantitative comparision, coming up with a representative performance evaluation criteria was difficult since not all the features offered by SPiKE is available on other frameworks. For example, no other existing framework except for Detours offers the ability to invoke the original construct at the instrumented location. JIT frameworks do not support instrumentation in kernel-mode and only a few of them allow instrumentation to be set at a function level in a clean fashion. (e.g. Pin does, but DynamoRIO does not). Also as mentioned before not all the JIT and probe-based frameworks are supported under various OSs. Thus, for our comparision we chose to measure only the intrument invocation time under the Linux OS for the frameworks (except for Detours). For Detours, we chose to compare both the instrument invocation time and the time for invoking the original construct at the instrumented location under the Windows OS. Considering that SPiKE scores over all the frameworks in terms of the features provided and the fact that our main aim is to show that the SPiKE's instrumentation achieves a low latency, the performance criteria we have chosen is acceptable.

Figure 6g shows the performance of SPiKE when compared to JIT frameworks. Since JIT frameworks are VM based, their instrument invocation time depends on the nature of the executing code. Hence, for our comparision we instrumented the file `stat` API for various applications under Linux and measured the normalized execution time of the applications both with and without instrumentation. The instrument for the file `stat` API had no processing whatsoever. This allowed us to measure and compare the instrument invocation time. As seen from the figure, SPiKEs instrumentation overhead is very minimal when compared to that of the JIT frameworks both with and without instrumentation.

Figure 6h shows the performance of SPiKE when compared to probe-based frameworks. The instrumentation latency of probe-based frameworks are not dependent on the nature of code that is executed. Hence, for our comparision we wrote a simple test application which made a single call to a file `open` API under the target OS and measured the latency in terms of clock cycles before the call and after the return. The instrument did nothing except to measure the latency of instrument invocation and invoking the original construct at the instrumented location (for frameworks that allow such a feature such as Detours and SPiKE). This allowed us to determine both the instrument invocation time as well as the time for the original construct invocation at the instrumented location. As seen from Figure 6h, SPiKE's performance is comparable to other probe-based frameworks, but is not the most efficient. Given that the other probe-based frameworks do not compare in capacity when it comes to malware analysis, the fact that SPiKE can achieve a latency close to these frameworks is acceptable.

6 Background and Related Work

There are various reseach related to the area of instrumentation and dynamic compilation. Broadly, instrumentation can be categorized as source level or binary level. Source level instrumentation basically includes insertion of wrappers in the program source code, which transfer control to the instrument during program execution. Binary level instrumentation on the other hand, accomplish instrumentation without the program sources. To limit our scope of discussion, we concentrate on binary level instrumentation in this section. Binary instrumentation can be categorized into static and dynamic approaches.

Static binary instrumentation is an offline technique that involves rewriting the program binary to insert instrumentation constructs. This art was pioneered by Atom (Srivastava & Eustace 1994), followed by others such as EEL (Larus & Schnarr 1995), Etch (Romer et al. 1997), Morph (Zhang et al. 1997) etc. Static approaches have a serious drawback in that, the tool may not have enough information to deal with mixed code and data within the executable. In the context of malware, static approaches do not suffice as most if not all malware are sensitive to code modification, being self-modifying and or self-checking. Other difficulties with static systems are indirect branches, dynamic libraries and dynamically generated code.

Dynamic binary instrumentation on the other hand involves inserting instrumentation during run-time, addressing the limitations of static approaches. There are two approaches to dynamic instrumentation: probe-based and JIT. Probe-based instrumentation works by dynamically replacing instructions in the target code, with instructions that branch to the instrumented code (jump or trap). Example probe-based frameworks include Dyninst (Buck & Hollingsworth 2000), Dtrace (Cantrill et al. 2004), Detours (Hunt & Brubacher 1999), DProbes (Moore 2001), Linux Trace Toolkit (Yaghmour & Dagenais 2000), Vulcan (Srivastava et al. 2001) etc. There are various drawbacks to probe-based approaches. In the context of malware, probe-based approaches have a severe limitation in that, they cannot cannot be used to probe malware specific functions since most if not all malware are very sensitve to code modification. Also such systems do not observe instrumentation transparency. With recent trend in malware showing increasing anti-analysis schemes, they can be easily detected and countered. Other problems with probe-based approaches are related to arbitrary construct instrumentation on architectures where instruction sizes vary (i.e x86). In such cases, an instruction cannot be replaced with one which is greater than its size, since it would overwrite the following instruction. JIT approaches on the other hand, overcome the transparency problem of probe-based approaches by executing code inside a VM. Examples include Pin (Luk et al. 2005), Valgrind (Nethercote & Seward 2003), DynamoRIO (Bruening 2004), Strata (Scott et al. 2003) and Diota (Maebe et al. 2002). However, in the context of malware, they do not support multithreading and do not carry support for SM-SC code. Also current JIT frameworks are unable to analyze code running in kernel-mode are very slow when compared to their probe-based counterparts. In comparison to the existing frameworks in the area of dynamic binary-instrumentation, SPiKE is unique in that, it is the first instrumentation framework specifically geared to aid in the construction of malware analysis tools. SPiKE provides dynamic coarse-grained binary-instrumentation that it is completely invisible to the target code and cannot be easily detected or countered. The framework has support for multithreading and SM-SC code and can capture code in user- and kernel-mode with minimal latency.

7 Conclusions

We have presented SPiKE, an unobtrusive, efficient, portable, easy-to-use and re-usable dynamic coarse grained binary-instrumentation framework for engineering malware analysis tools. The instrumentation deployed by the framework is completely invisible to the target code and cannot be easily detected or countered. We show that the framework can capture multithreaded and SM-SC code in both user- and kernel-mode while incurring a minimal performance latency. SPiKE currently runs under the Windows and Linux OSs on IA-32 (and compatible) processors. The framework achieves instrumentation using the virtual memory system as a base, that is a commonplace in most platforms. This, coupled with the fact that the SPiKE architecture abstracts platform specific details, enables the framework to be ported to other platforms. We show SPiKEs easy to use APIs enable construction of powerful malware analysis tools with ease and discuss one of our own tools that we have used for behavior monitoring of various malware.

Although there remain other important features of SPiKE for which space does not permit a detailed description (selective instrumentation, support APIs, slicing internals, framework polymorphism etc.), we have shown how SPiKE addresses the shortcomings in current research involving binary-instrumentation in the context of malware. In our belief, the framework is the first of its kind in tailoring an instrumentation strategy conducive to malware analysis. Future works include: (a) raising the stealth levels by developing a supervised code execution environment for privileged malware code, (b) employ a sophisticated code rewriter which would do away with the need for redirection-pads and (c) integrate SPiKE into a full fledged malware analysis environment currently being developed by us.

References

Buck, B. R. & Hollingsworth, J. (2000), An api for runtime code patching, *in* 'Journal of High Performance Computing Applications', Vol. 14(4), pp. 317-329.

Bruening, D. L. (2004), Efficient, Transparent, and Comprehensive Runtime Code Manipulation, Ph.D., M.I.T. (http://www.cag.lcs.mit.edu/dynamorio/).

Cantrill, B. M., Shapiro, M. W. & Leventhal, A. H. (2004), Dynamic instrumentation of production systems, *in* '6th Symposium on Operating Systems Design and Implementation', pp. 15–28.

Hunt, G. & Brubacher, D. (1999), Detours: Binary Interception of Win32 Functions, *in* '3rd USENIX Windows NT Symposium', pp. 135–144.

Intel Corporation. (2003), *IA-32 Intel Architecture Software Developers Manual.*, Vols 1-3.

Larus, J. & Schnarr, E. (1995), EEL: Machine-independent executable editing, *in* 'ACM SIGPLAN Conference on Programming Language Design and Implementation', pp. 291-300.

Luk, C., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J. & Hazelwoo, K. (2005), Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation, *in* 'ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)', Chicago, IL, USA, pp. 190–200.

Maebe, J., Ronsse, M. & De Bosschere, K. (2002), Diota: Dynamic instrumentation, optimization and transformation of applications, *in* 'Compendium of Workshops and Tutorials held in conjunction with PACT02'.

Microsoft Corporation. (2004), Microsoft Portable Executable and Common Object File Format Specification., Rev.

Moore, R. J. (2001), A universal dynamic trace for Linux and other operating systems, *in* 'FREENIX Track'.

Nethercote, N. & Seward, J. (2003), Valgrind: A program supervision framework, *in* '3rd Workshop on Runtime Verification'.

Romer, T., Voelker, G., Lee, D., Wolman, A., Wong, W., Levy, H., Bershad, B. & Chen. B. (1997), Instrumentation and optimization of win32/intel executables using Etch, *in* 'USENIX Windows NT Workshop', pp. 1-7.

Scott, K., Kumar, N., Velusamy, S., Childers, B., Davidson, J. & Soffa, M. L. (2003), Reconfigurable and retargetable software dynamic translation, *in* '1st Conference on Code Generation and Optimization', pp. 36-47.

Srivastava, A., Edwards, A. & Vo, H. (2001), Vulcan: Binary transformation in a distributed environment, Technical Report MSR-TR-2001-50, Microsoft Research.

Srivastava, A. & Eustace, A. (1994), Atom: A system for building customized program analysis tools, *in* 'ACM SIGPLAN Conference on Programming Language Design and Implementation', pp. 196-205.

Unix System Laboratories. (1998), Tool Interface Standards. ELF: Executable and Linkable Format.

Vasudevan, A. & Yerraballi, R. (2005), Stealth Breakpoints, *To appear in* '21st Annual Computer Security and Applications Conference', Tucson, AZ, USA.

Vasudevan, A. & Yerraballi, R. (2004), SAKTHI: A Retargetable Dynamic Framework for Binary Instrumentation, *in* 'Hawaii International Conference in Computer Science', Honolulu, HI, USA.

Yaghmour, K. & Dagenais, M. R. (2000), Measuring and characterizing system behavior using kernel-level event logging, *in* 'USENIX Annual Technical Conference', pp. 13–26.

Zhang, X., Wang, Z., Gloy, N., Chen, J. B. & Smith, M. D. (1997), System support for automatic profiling and optimization, *in* '16th Symposium on Operating System Principles', pp. 15–26.

A framework for role-based group delegation in distributed environments

Hua Wang Jiuyong Li Ron Addie Stijn Dekeyser Richard Watson

Department of Maths & Computing, University of Southern Queensland
Toowoomba QLD 4350 Australia
Email: `(wang, jiuyong, addie, dekeyser, rwatson)@usq.edu.au`

Abstract

Role-based delegation model (*RBDM*) based on the role-based access control (*RBAC*) has proven to be a flexible and useful access control model for information sharing in a distributed collaborative environment. In today's highly dynamic distributed systems, a user often needs to delegate a role to all members of a group at the same time. It presents the challenge of how to build a role-based group delegation framework within *RBAC* in distributed environment.

This paper aims to build a group delegation framework within *RBAC*. The framework includes a role-based group delegation granting model, group delegation revocation model, granting authorization and revocation authorization. We analyze various revocations and the impact of revocations on role hierarchies. The implementation with *XML* based tools demonstrates the framework and authorization methods. Finally, comparisons with other related work are indicated.

1 Introduction

The National Institute of Standards and Technology developed the role-based access control (*RBAC*) prototype (Feinstein H. L. 1995) and published a formal model (Ferraiolo D. F. and Kuhn D. R. 1992). *RBAC* has been widely used in database system management and distributed environments since it enables managing and enforcing security in large-scale and enterprise-wide systems. *RBAC* involves individual users being associated with roles as well as roles being associated with permissions (Each permission is a pair of objects and operations). As such, a role is used to associate users and permissions. A user in this model is a human being. A role is a job function or job title within the organization associated with authority and responsibility.

Permission is an approval of a particular operation to be performed on one or more objects. As shown in Figure 1, the relationships between users and roles, and between roles and permissions are many-to-many (i.e. a permission can be associated with one or more roles, and a role can be associated with one or more permissions). The security policy of the organization determines role membership and the allocation of each role's capabilities.

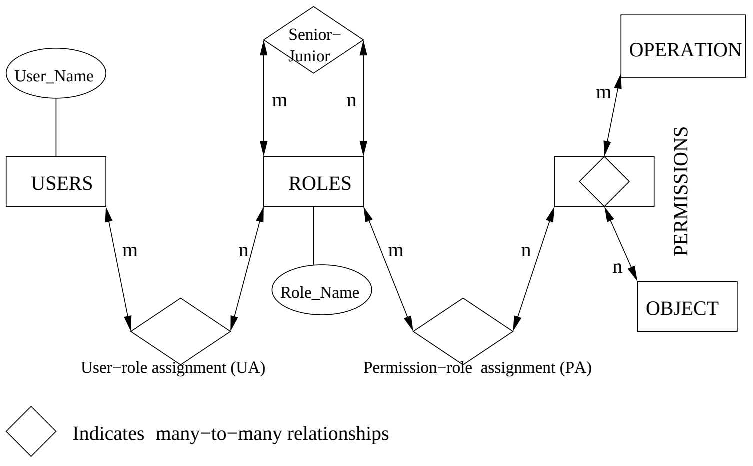
The *RBAC* model supports the specification of several aspects.

a. User/role associations – the constraints specifying user authorizations to perform roles;
b. Role hierarchies – the constraints specifying which role may inherit all of the permissions of another role;
c. Duty separation constraints – these are role/role associations indicating conflict of interest:
c1. Static separated duty (*SSD*) – a constraint specifying that a user cannot be authorized for two different roles;
c2. Dynamic separated duty (*DSD*) – a constraint specifying that a user can be authorized for two different roles but cannot act simultaneously in both;
d. Cardinality – the maximum number of users allowed, i.e. how many users can be authorized for any particular role (role cardinality), e.g., only one manager.

The number of roles and users in a large enterprise system can be hundreds or thousands. Managing these roles and users, and their interrelationships is a vital challenge that is often highly decentralized and delegated to a small team of project groups. User-role assignment is a particularly critical administrative activity because assigning people to tasks is a normal managerial function and assigning users to roles is a natural part of assigning users to tasks. Furthermore, the activities in distributed environment are decentralized and delegated to users rather than system administrators.

Delegation is an important aspect of *RBAC* and often regarded as one of the principal motivation behind *RBAC*. Although the importance of delegation in *RBAC* has been recognozed for a long time, it has not received much attention. Zhang et al (Zhang L., Ahn G., and Chu B. 2001, Zhang L., Ahn G., and Chu B. 2002) recently proposed a rule-based framework for role-based delegation including the *RDM2000* model. We use the concept of role-based delegation borrowed from their work. The central contributions of this article are to describe how we can build a framework of role-based group delegation within *RBAC* in a distributed environment and to implement the delegation framework with *XML* based tools and languages.

The remainder of this paper is organized as follows: Section 2 presents the related work associated with the delegation models and *RBAC*. As the results of this section, we find that both group-based delegation within *RBAC* and its implementation with *XML* has not been analysed and presented in the literature. Section 3 proposes a delegation framework which includes the structures of role-based delegation and role-based group delegation models. Section 4 provides delegation authorizations. Granting authorization with pre-requisite conditions and revocation authorization are discussed in this section. Definitions of *Can_delegate*, *Can_revoke*, *role range* are introduced. Section 5 describes the implementation of



Senior−Junior
User_Name
OPERATION
m
n
m
USERS
ROLES
PERMISSIONS
m
n
m
n
n
Role_Name
OBJECT
User−role assignment (UA)
Permission−role assignment (PA)
Indicates many−to−many relationships

Figure 1: RBAC relationship.

the role-based group delegation using *XML* technology and Section 6 compares the work in this paper and related previous work. Finally, the conclusion of the paper is in Section 6.

2 Related work

Delegation is an important feature in many collaboration applications. For example, imagine that the Immigration Department is developing partnerships between immigration agencies and people in local areas to address possible problems. Immigration officers are able to prevent illegal stays and crime if they efficiently collaborate with the people. The problem-oriented immigrating system (*POIS*) is proposed to improve the service as a part of the Immigration Department's ongoing community efforts including identifying potential problems and resolving them before they become significant. With efficient delegation, officers respond quickly to urgent messages and increase the time spent confronting problems.

In *POIS*, officers might be involved in many concurrent activities such as conducting initial investigations, analysing and confronting crimes, preparing immigration reports, and assessing projects. In order to achieve this, users may have one or more roles such as lead officer, participant officer, or reporter. In this example, Tony, a director, needs to coordinate analysing and confronting crimes and assessing projects. Collaboration is necessary for information sharing with members from these two projects. To collaborate closely and make two projects more successful, Tony would like to delegate certain responsibilities to Christine and her staff. The prerequisite conditions are to secure these processes and to monitor the progress of the delegation. Furthermore, Christine may need to delegate the delegated role to her staff as necessary or to delegate a role to all members of a group at the same time. Without delegation skill, security officers have to do excessive work because of their involvement in every single collaborative activity. The major requirements of role-based delegation in this example are:

1. Group-based delegation means that a delegating user may need to delegate a role to all members of a group at the same time.
2. Multistep delegation occurs when a delegation can be further delegated. Single-step delegation means that the delegated role cannot be further delegated.
3. Revocation schemes are an important feature of collaboration systems. They take away the delegated permissions. There are different revoking schemes; among them are strong and weak revocations, cascading and noncascading revocations, as well as grant-dependent and grant-independent revocations (Wang H., Cao J. and Zhang Y. 2003).
4. Constraints are an important factor in *RBAC* for laying out higher-level organizational policies (Wang H., Cao J. and Zhang Y. 2001). They define whether or not the delegation or revocation process is valid.
5. Partial delegation means only subsets of the permissions are delegated while total delegation means all permissions are delegated. Partial delegation is an important feature because it allows users only to delegate required permissions. The well-known least privilege security principle can be implemented through partial delegation.

Although the concept of delegation is not new in authorizations (Aura T. 1999, Barka E. and Sandhu R. 2000a, Wang H., Zhang Y., Cao J. and Varadharajan V. 2003, Wang H., Sun L., Zhang Y., and Cao J. 2005), role-based delegation received attention only recently (Barka E. and Sandhu R. 2000a, Barka E. and Sandhu R. 2000b, Zhang L., Ahn G., and Chu B. 2001, Zhang L., Ahn G., and Chu B. 2002). Aura (Aura T. 1999) introduced key-oriented discretionary access control systems that are based on delegation of access rights with public-key certificates. The systems emphasized decentralization of authority and operations but their approach is a form of discretionary access control. Hence, they can neither express mandatory policies like the Bell-LaPadula model (Bell D.E., La Padula L.J. 1976), nor is it possible to verify that someone does not have a certificate. Furthermore, some important policies such as separation of duty policies cannot be expressed with only certificates. They need some additional mechanism to maintain the previously granted rights and the histories must be updated in real time when new certificates are issued. Delegation is also applied in decentralized trust management (Blaze M. Feigenbaum J., Ioannidis J. and Keromytis A. 1999, Li N. and Grosof B. N. 2000). Blaze et al (Blaze M. Feigenbaum J., Ioannidis J. and Keromytis A. 1999) identified the trust management problem as a distinct and important component of security in network services and Li et al (Li N. and Grosof B. N. 2000) made a logic-based knowledge representation for authorization with tractable trust-management in large-scale, open, distributed systems. Delegation was used to address the trust management problem including formulating security policies and security credentials, determining whether particular sets of credentials satisfy the relevant policies, and deferring trust to third parties. Other researchers have investigated machine to machine and human to machine delegations (Wang H., Cao J. and Zhang Y. 2001, Abadi M., Burrows M., Lampson B., and Plotkin G. 1993). For example, Wang et al (Wang H., Cao J. and Zhang Y. 2001) proposed a secure, scalable anonymity payment protocol for Internet purchases through an agent which provided a higher anonymous certificate and improved the security of consumers. The agent certified re-encrypted data after verifying the validity of the content from consumers. The agent is a human to machine delegation which can provide new certificates. However, many important role-based concepts, for example, role hierarchies, constraints, revocation were not mentioned.

Zhang et al (Zhang L., Ahn G., and Chu B. 2001, Zhang L., Ahn G., and Chu B. 2002) proposed a rule-based framework for role-based delegation including the *RDM2000* model. The *RDM2000* model is based on the *RBDM0* model which is a simple delegation model supporting only flat roles and single step delegation. Furthermore, as a delegation model, it does not support group-based delegation.

This paper focuses exclusively on a role-based delegation model which supports group-based delegation and its implementation with *XML* technology. We extend our previous work and propose a delegation framework including delegation granting and revocation models, group-based delegation. To provide sufficient functions with the framework, this paper analyses how changes to original role assignment impact upon delegation results. This kind of role-based group delegation and its implementation with *XML* have not been studied before.

3 Role-based group delegation framework

In this section we propose a role-based group delegation framework called *RBGDF* which supports role hierarchy and group delegation by introducing the delegation relation.

3.1 Role-based delegation model

A session is an important concept within *RBAC* which means a mapping between a user and possibly many roles. For example, a user may establish a session by activating some subset of assigned roles. A session is always associated with a single user and each user may establish zero or more sessions. There may be hierarchies within roles. Senior roles are shown at the top of the hierarchies. Senior roles inherit permissions from junior roles. Let $x > y$ denote x is senior to y with obvious extension to $x \geq y$. Role hierarchies provide a powerful and convenient means to enforce the principle of least privilege since only required permissions to perform a task are assigned to the role.

Although the concept of a user can be extended to include intelligent autonomous agents, machines, even networks, we limit a user to a human being in our model for simplicity.

Figure 2 shows the role hierarchy structure of *RBAC* in *POIS*. The following Table 1 expresses an example of user-role assignment in *POIS*.

There are two sets of users associated with role r:

Original users are those users who are assigned to the role r;

RoleName	UserName
DIR	Tony
HO1	Christine
HO2	Mike
Co1	Richard
Re1	John
CS	Ahn

Table 1: User-Role relationship.

Delegated users are those users who are delegated to the role r.

The same user can be an original user of one role and a delegated user of another role. Also it is possible for a user to be both an original user and a delegated user of the same role. For example, if Christine delegates her role *HO1* to Richard, then Richard is both an original user (explicitly) and a delegated user (implicitly) of role *Co1* because the role *HO1* is senior to the role *Co1*. The original user assignment (UAO) is a many-to-many user assignment relation between original users and roles. The delegated user assignment (UAD) is also a many-to-many user assignment relation between delegated users and roles.

We have the following components for role-based delegation model:

U, R, P and S are sets of users, roles, permissions, and sessions, respectively.

1. $UAO \subseteq U \times R$ is a many-to-many original user to role assignment relation.

2. $UAD \subseteq U \times R$ is a many-to-many delegated user to role assignment relation.

3. $UA = UAO \cup UAD$.

4. Users: $R \Rightarrow 2^U$is a function mapping of roles to sets of users.
   $Users(r) = \{u|(u, r) \in UA\}$ where UA is user-role assignment.

5. $Users(r) = Users\_O(r) \cup Users\_D(r)$
   where
   $Users\_O(r) = \{u|\exists r' \geq r, (u, r') \in UAO\}$
   $Users\_D(r) = \{u|\exists r' \geq r, (u, r') \in UAD\}$

   $Users(r)$ includes all users who are members of role r. The users may be original and delegated users. The original users $Users\_O(r)$ are not only the member of role r but also the member of a senior role of r. The members in $Users\_D(r)$ are similar to that in $Users\_O(r)$.

With these components, we analyse group delegation in the remaining part of this section.

3.2 Role-based group Delegation

The scope of our model is to address user-to-user delegation supporting role hierarchies and group delegations. We consider only the regular role delegation in this paper, even though it is possible and desirable to delegate an administrative role.

A delegation relation (*DELR*) exists in the role-based delegation model which includes three elements: original user assignments *UAO*, delegated user assignment *UAD*, and constraints. The motivation behind this relation is to address the relationships among different components involved in a delegation. In a user-to-user delegation, there are five components: a delegating user, a delegating role, a delegated user, a delegated role, and associated constraints.

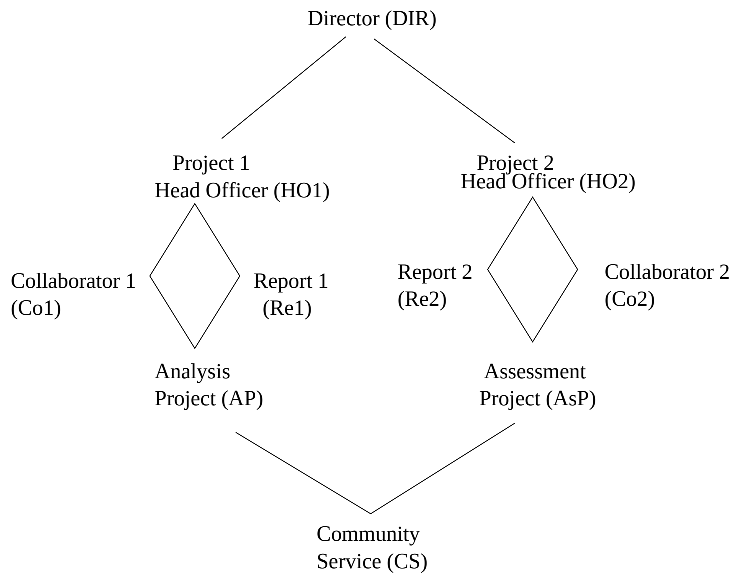


Figure 2: Role hierarchy in *POIS*.

For example, *((Tony, DIR), (Christine, DIR), Friday)* means Tony acting in role *DIR* delegates role *DIR* to Christine on Friday. We assume each delegation is associated with zero or more constraints. The delegation relation supports partial delegation in a role hierarchies: a user who is authorized to delegate a role r can also delegate a role r' that is junior to r. For example, *((Tony, DIR), (Ahn, Re1), Friday)* means Tony acting in role *DIR* delegates a junior role *Re1* to Ahn on Friday. A delegation relation is one-to-many relationship on user assignments. It consists of original user delegation (*ORID*) and delegated user delegation (*DELD*). Figure 3 illustrates components and their relations in a role-based delegation model.

From the above discussions, the following components are formalized:

1. $DELR \subseteq UA \times UA \times Cons$ is one-to-many delegation relation. A delegation relation can be represented by $((u, r), (u', r'), Cons) \in DELR$, which means the delegating user u with role r delegated role r' to user u' when the constraint $Cons$ is satisfied.

2. $ORID \subseteq UAO \times UAD \times Cons$ is an original user delegation relation.

3. $DELD \subseteq UAD \times UAD \times Cons$ is a delegated user delegation relation.

4. $DELR = ORID \cup DELD$

The last equation shows that delegation relations consist of original and delegated user delegation relations.

Now we analyse group delegation. In this paper we only discuss user-group delegations which consist of original user-group and delegated user-group delegations. The new relation of group delegation is defined as delegation group relation (*DELGR*) which includes: original user assignments *UAO*, delegated user assignments *UAD*, delegated group assignments *GAD*, and *constraints*. In a user-group delegation, there are five components: a delegating user (or a delegated user), a delegating role, a delegated group, a delegated role, and associated constraints. For example, *((Tony, DIR), (Project 1, DIR), 1:00pm–3:00pm Monday)* means Tony acting in role *DIR* delegates role *DIR* to all people involved in Project 1 during 1:00pm–3:00pm on Monday. A group delegation relation is one-to-many relationship on user assignments. It consists of original user group delegation (*ORIGD*) and delegated user group delegation (*DELGD*). Figure 4 illustrates components and their relations in role-based delegation model. Hence we have the following elements and functions in group delegation:

1. G is a set of users. GA is a set of group-role assignments.

2. $DELGR \subseteq UA \times GA \times Cons$ is one-to-many delegation relation. A delegation relation can be represented by $((u, r), (G, r), Cons) \in DELR$, which means the delegating user u with role r delegated role r to group G if the constraint $Cons$ is satisfied.

3. $ORIGD \subseteq UAO \times GAD \times Cons$ is a relation of an original user and a group with constraints.

4. $DELGD \subseteq UAD \times GAD \times Cons$ is a relation of a delegated user and a group with constraints.

5. $DELGR = ORIGD \cup DELGD$.

Based on the results of the structure with role-based group delegation, we discuss group delegation authorizations in the next section.

4 Delegation Authorization

We develop delegating and revocation models in this section. The notion of a *prerequisite condition*, $Can\_delegate$ and Can_revoke are key parts in group delegation process.

4.1 Authorization models

The delegation authorization goal imposes restrictions on which role can be delegated to whom. We partially adopt the notion of prerequisite condition from (Wang H., Cao J. and Zhang Y. 2003) to introduce delegation authorization in the delegation framework.

A *prerequisite condition* is an expression using Boolean operators $\wedge$ and $\vee$ on terms of the form r and $\bar{r}$ where r is a role and $\wedge$ means "and", $\vee$ means "or". A prerequisite condition is evaluated for a user u by interpreting r to be true if $(\exists r' \geq r), (u, r') \in UA$ and $\bar{r}$ to be true if $(\forall r' \geq r), (u, r') \notin UA$, where UA is a set of user-role assignments. ⋄

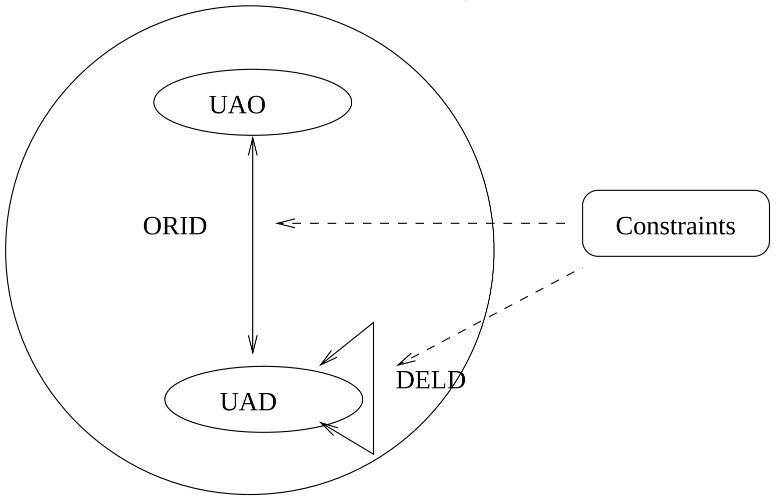


Figure 3: Role-based delegation model.

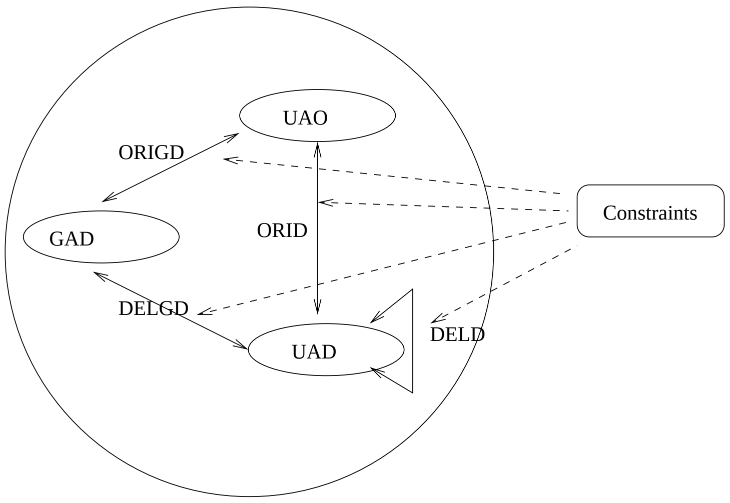


Figure 4: Role-based group delegation model.

We say a group satisfies a prerequisite condition if all users in the group satisfy the prerequisite condition.

For a given set of roles R let CR denote all possible prerequisite conditions that can be formed using the roles in R, for example, $CR = r_1 \wedge r_2 \vee \bar{r}_3$. In some cases, we may need to define whether or not a user can delegate a role to a group and for how many times, or up to the maximum delegation depth. Not every user can delegate a role to a user. The following relation provides what roles a user can delegate with prerequisite conditions.

Definition 1 *Can_delegate* is a relation of $R \times CR \times N$ where R, CR, N are sets of roles, prerequisite conditions, and maximum delegation depth, respectively. ◇

The meaning of $(r, cr, n) \in Can\_delegate$ is that a user who is a member of role r (or a role senior to r) can delegate role r (or a role junior to r) to any group whose current entitlements in roles satisfy the prerequisite condition CR without exceeding the maximum delegation depth n. To identify a role range within the role hierarchy, the following closed and open interval notation is used.

$$[x, y] = \{r \in R | x \geq r \wedge r \geq y\}$$
$$(x, y] = \{r \in R | x > r \wedge r \geq y\}$$
$$[x, y) = \{r \in R | x \geq r \wedge r > y\}$$
$$(x, y) = \{r \in R | x > r \wedge r > y\}$$

User-group delegation is authorized by *Can_delegate*. Table 2 shows the *Can_delegate* relations with the prerequisite conditions in the *POIS* example. The meaning of *Can_delegate* $(DIR, [CS, HO1], 1)$ is that a member of role *DIR* can delegate role *DIR* and all roles in *POIS* (since all roles are junior to *DIR*) to a group whose current membership satisfies the prerequisite condition *[CS, HO1]* with one-step delegation. The second tuple authorizes that a user of role *HO1* can assign role *HO1*, and *Co1*, *Re1*, *AP* and *CS* to a group in which users are members of either role *AP*, or *Co1*, or *Re1* (since *AP*, *Co1* and *Re1* are in the range of *[AP, HO1))*.

RoleName (R)	Prereq.Condition (CR)	N
DIR	[CS, HO1]	1
HO1	[AP, HO1)	2
AP	CS	1
CS	ϕ	2

Table 2: Can delegate relations in POIS.

There are related subtleties that arise in *RBGDF* concerning the interaction between delegating and revocation of user-group delegation membership and the role hierarchy.

Definition 2 A user-group delegation revocation is a relation $Can\_revoke \subseteq R \times 2^R$, where R is the set of roles. ⋄

The meaning of $Can\_revoke(x, Y)$ is that a member of role x (or a member of a role that is senior to x) can revoke delegation relationship of a group from any role $y \in Y$, where Y defines the *range of revocation.* Table 3 gives the Can-revoke relation in Figure 2. The first tuple shows that a member of role *HO1* can revoke a delegation relationship of a group from any role in *[Co1, CS]*.

RoleName	Role Range
HO1	[Co1, CS]
Re1	[Re1, AP]

Table 3: Example of can revoke relation.

There are two kinds of revocations (Wang H., Cao J. and Zhang Y. 2003). The first one is weak revocation while the second one is strong revocation. We extend the definition of explicit and implicit members of a role from a user to a group.

Definition 3 A group G is an explicit member of a role x if $(u, x) \in UA$, for all $u \in G$, and that G is an implicit member of role x if for some $x' > x, (u, x') \in UA$, for all $u \in G$. ⋄

Weak revocation only revokes explicit membership from a user and does not revoke implicit membership. On the other hand, strong revocation requires revocation of both explicit and implicit membership. Strong revocation of $G's$ membership in x requires that G be removed not only from explicit membership in x, but also from explicit (implicit) membership in all roles senior to x. Strong revocation therefore has a cascading effect upwards in the role hierarchy. For example, suppose there are two delegations $((Tony, DIR), (Ahn, AP), Friday)$ and $((John, Re1), (Ahn, AP), Friday)$ and Tony wants to remove the membership of AP from Ahn on Friday. With weak revocation, the first delegation relationship is removed, but the second delegation has not yet removed. It means that Ahn is still a member of AP. With strong revocation two delegation relationships are removed and hence Ahn is not a member of AP.

5 XML implementation

This section presents the implementation of the group delegation with *XML* technology. The format of a group delegation from Section 3 is $((u, r), (G, r), Cons)$. To maintain the relationship between groups, we extend the definition of senior and junior role to the definition of senior and junior group.

Definition 4 A group $G1$ is senior to a group $G2$ if any member of $G1$ has the power of the member in $G2$ and may have additional power but not vice versa. ⋄

Let $G1 > G2$ signify that $G1$ is senior to $G2$. Hence a member of $G1$ is considered senior to a member of $G2$. If $G1$ is senior to $G2$ we also say that $G2$ is junior to $G1$. For convenience we use the files *grouphie.xml* and *rolehie.xml* to store the group hierarchy and the role hierarchy of the children and parents groups and roles.

Based on Table 1, a part of the group hierarchy of Figure 2 is modelled in *grouphie.xml* using *IDREF* attibutes (Michael H. 2001) as shown in Table 4. The hierarchy is not a tree but a graph, for clarity and conciseness, Table 4 shows the hierarchy as a nested relation. The first column gives the group name, the second column gives the immediate parent groups of that group, and the third column gives the immediate children. The ϕ means that the group has no parent or child as the case may be. Using *grouphie.xml*, we can find all seniors and juniors for a group by respectively chasing the parents and children using simple *XPath* query expressions. An example of the role hierarchy of Figure 2 is represented in *rolehie.xml* using *IDREF* attibutes as shown in Table 5.

Role Name	Senior role	Junior role
DIR	ϕ	HO1, HO2
HO1	DIR	Co1, Re1
HO2	DIR	Co2, Re2
Co1	HO1	AP
Re1	HO1	AP
AP	Co1, Re1	CS
CS	AP, AsP	ϕ

Table 5: Group hierarchy of Figure 2.

Definition 3 has described a group to be an *explicit* or *implicit* member of a role. A group may be an *explicit* and *implicit* member of a role simultaneously. To simulate a role hierarchy we use information about explicit and implicit membership in *roleDB*. However, *roleDB* is not sufficient to distinguish the case where a group is both an explicit and implicit member of some role from the case where the group is only an implicit member of the role. For this purpose we introduce another file *explicit.xml* that keeps information about explicit membership only.

There is a procedure for delegating a user to a group in our implementation. The procedure call is *Delegate (role, group)*. The parameters role and group specify which role is to be delegated to a group. The delegation function has the following main steps: 1) Select a role to be delegated (or revoked); 2) Select a group to delegate (or revoke); 3) Check whether ro not the group satisfies the prerequisite condition in the relation *Can_deleagte*; 4) Setup constraints; and 5) Update the *DELGR* database. For delegation revocation, instead of the steps 3 and 4, check whether or not the revoked role is in the role range of the relation Can_revoke. The *DELGR* database maintains group hierarchy information (*grouphie.xml*), role hierarchy information (*rolehie.xml*), explicit membership (*explicit.xml*), and the $Can\_delegate$ and Can_revoke relation tables.

In order to make our implementation more convenient we developed a graphical user interface which interacts with this procedure to do role-based group delegation. The graphical user interface is illustrated in Figure 5. This interface was developed using *XUL* and is used to initiate group delegation instead of typing the above procedure call. This implementation is convenient for users since they only need to define the group hierarchy and the relation can-assign.

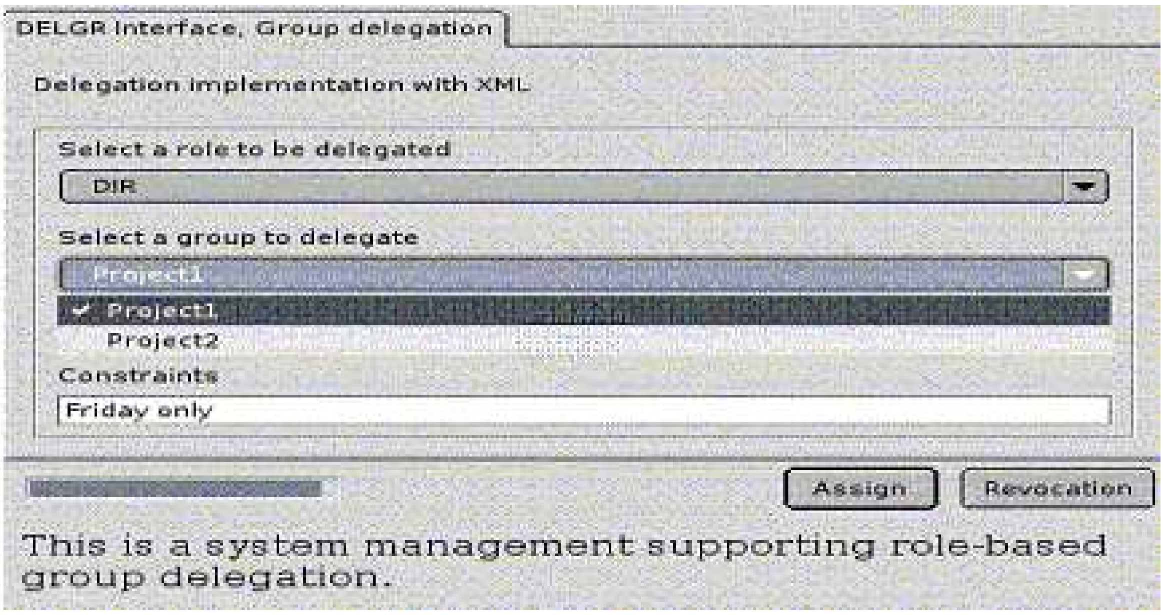
6 Comparisons

The closed work to this paper is on mobility of user-role assignment (Wang H., Sun L., Zhang Y., and Cao J. 2005) and role-based delegation (Barka E. and Sandhu R. 2000a).

Our previous work (Wang H., Sun L., Zhang Y., and Cao J. 2005) discussed the mobility of user-role relationship in *RBAC* management and provided new authorization allocation algorithms for *RBAC* along with mobility that are based on relational algebra operations. They are the authorization granting algorithm, weak revocation algorithm and strong revocation algorithm. The paper does not use role delegation but instead defines the role mobility, whereby a user with an mobile role may further grant other

Group Name	Parent group	Child group
Tony	ϕ	Project1, Project2
Project1	Tony	Ahn
Project2	Tony	Ahn
Ahn	Project1, Project2	ϕ

Table 4: Group and its roles hierarchy of Figure 2.



DELGR Interface, Group delegation

Delegation implementation with XML

Select a role to be delegated: DIR

Select a group to delegate: Project1 (✓ Project1, Project2)

Constraints: Friday only

This is a system management supporting role-based group delegation.

Figure 5: Group delegation interface.

roles but she/he cannot accept other roles if she/he has an immobile role. The mobility could be viewed as a special case of role-based delegation in their work. But some important delegation features such as delegation conflicts and delegation revocation have not been considered. By contrast, the work in this paper provides a rich variety of options that can deal with delegation authorization and revocation.

Barka and Sandhu (Barka E. and Sandhu R. 2000a) proposed a simple model for role-based delegation called *RBDM0* within *RBAC0*, the simplest form of *RBAC*96 (Sandhu R. 1997). They developed a framework for identifying interesting cases that can be used for building role-based delegation models. This is accomplished by identifying the characteristics related to delegation, using these characteristics to generate possible delegation cases. Their work is different from ours in three aspects. First, it focuses on a simple delegation model supporting only flat roles and single step delegation. Some important features such as role hierarchies, constraints and revocations were not supported. By contrast, our work has analysed delegation authorization and revocation models with constraints involving role hierarchies. Second, they neither gave the definition of role-based delegation relation, which is a critical notion to the delegation model nor discussed the relationships among original user and delegated user. By contrast, the delegation framework in this paper is based on original user and delegated user since the delegating relationship in this paper has five components $((u,r),(u',r'),Cons)$. Third, they have not discussed group-based delegation, but we have analysed elements and functions in group delegation as well as its implementation with *XML*.

7 Conclusions

This paper has discussed a role-based delegation model and its implementation with *XML*. We have analysed not only a delegating framework including delegating authorization and revocation with constraints, but also group-based delegation. To provide a practical solution for role-based group delegation, we have analysed role hierarchies and the relationship of senior and junior roles. The theory in this paper was demonstrated by its implementation with *XML*. The work in this paper has significantly extended previous work in several aspects, for example, the group-based delegation, group delegation authorization with prerequisite conditions and revocation authorization.

The future work will be the development of a system management with *XML* which involves the role-based group delegation subsystem.

Acknowledgment

The authors would like to thank reviewers for their good suggestions and comments.

References

Abadi M., Burrows M., Lampson B., and Plotkin G. (1993), 'A calculus for access control in distributed systems', *ACM Trans. Program. Lang. Syst.* **15**(4), 706–734.

Aura T. (1999), 'Distributed access-rights management with delegation certificates', *Security Internet programming* pp. 211 – 235.

Barka E. and Sandhu R. (2000a), Framework for role-based delegation models and some extensions, *in* ‘Proceedings of the 16 Annual Computer Security Applications Conference’, New Orleans, pp. 168–177.

Barka E. and Sandhu R. (2000b), Framework for role-based delegation models, *in* ‘Proceedings of the 23rd National Information Systems Security Conference’, Baltimore, pp. 101–114.

Bell D.E., La Padula L.J. (1976), ‘Secure computer system: Unified exposition and multics interpretation’, Technical report ESD-TR-75-306.

Blaze M. Feigenbaum J., Ioannidis J. and Keromytis A. (1999), ‘The role of trust management in distributed system security’, *Security Internet programming* pp. 185 – 210.

Feinstein H. L. (1995), Final report: Nist small business innovative research (sbir) grant: role based access control: phase 1. technical report, *in* ‘SETA Corp.’.

Ferraiolo D. F. and Kuhn D. R. (1992), Role based access control, *in* ‘15th National Computer Security Conference’, ferraiolo92rolebased.html, pp. 554–563.

Li N. and Grosof B. N. (2000), A practically implementation and tractable delegation logic, *in* ‘IEEE Symposium on Security and Privacy’, pp. 27–42.

Michael H. (2001), *XSLT Programmer’s Reference*, Wiley.

Sandhu R. (1997), Rational for the *RBAC*96 family of access control models, *in* ‘Proceedings of 1st ACM Workshop on Role-based Access Control’, ACM Press, pp. 64–72.

Sandhu R. (1998), Role activation hierarchies, *in* ‘Third ACM Workshop on RoleBased Access Control’, ACM Press, pp. 33–40.

Wang H., Cao J. and Zhang Y. (2001), A consumer anonymity scalable payment scheme with role based access control, *in* ‘2nd International Conference on Web Information Systems Engineering (WISE01)’, Kyoto, Japan, pp. 53–62.

Wang H., Cao J. and Zhang Y. (2003), Formal authorization allocation approaches for permission-role assignments using relational algebra operations, *in* ‘Proceedings of the 14th Australian Database Conference ADC2003’, Adelaide, Australia.

Wang H., Cao J., Zhang Y. (2005), ‘A flexible payment scheme and its role based access control’, *IIEEE Transactions on Knowledge and Data Engineering* **17**(3), 425–436.

Wang H., Sun L., Cao J., and Zhang Y. (2004), Anonymous access scheme for electronic-services , *in* ‘Proceedings of the Twenty-Seventh Australasian Computer Science Conference (ACSC2004)’, Dunedin, New Zealand, pp. 296–305.

Wang H., Sun L., Zhang Y., and Cao J. (2005), Authorization Algorithms for the Mobility of User-Role Relationship, *in* ‘Proceedings of the 28th Australasian Computer Science Conference (ACSC2005)’, Newcastle, Australia, pp. 167–176.

Wang H., Zhang Y., Cao J. and Kambayahsi Y. (2004), ‘A global ticket-based access scheme for mobile users’, *Special Issue on Object-Oriented Client/Server Internet Environments, Information Systems Frontiers* **6**(1), 35–46.

Wang H., Zhang Y., Cao J. and Varadharajan V. (2003), ‘Achieving secure and flexible m-services through tickets’, *IEEE Transactions on Systems, Man, and Cybernetics, Part A, Special issue on M-Services* pp. 697–708.

Yao W., Moody K. and Bacon J. (2001), A model of oasis role-based access control and its support for active security, *in* ‘Proceedings of ACM Symposium on Access Control Models and Technologies’, pp. 171–181.

Zhang L., Ahn G., and Chu B. (2001), A rule-based framework for role-based delegation, *in* ‘Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT 2001)’, Chantilly, VA, pp. 153–162.

Zhang L., Ahn G., and Chu B. (2002), A role-based delegation framework for healthcare information systems, *in* ‘Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT 2002)’, Monterey, CA, pp. 125–134.

Author Index